

SIEMENS EDA

Calibre® YieldAnalyzer and YieldEnhancer User's and Reference Manual

Software Version 2021.2
Document Revision 15

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Unpublished work.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
15	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
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 January 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 December 2020
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

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 DFM Tools		17
Calibre DFM Overview		17
Calibre DFM WorkFlow		18
Key Concepts		20
Calibre YieldAnalyzer		20
Calibre YieldEnhancer		21
Calibre YieldServer		24
Calibre DFM Requirements.....		25
TVF Requirements.....		25
Syntax Conventions		26
Chapter 2		
Getting Started with Calibre DFM		29
Calibre DFM Command Line Invocation		30
Calibre Command Line Modes.....		30
calibre		32
Invocation of Calibre DFM from Calibre Interactive		36
DFM Design Environment Management.....		36
DFM Rule File Basics		37
Calibre DFM Output Database Format		39
Saving DFM Results to a DFM Database.....		39
Saving DFM Results to an RDB.....		40
DFM-Specific SVRF Rule File Entries		43
Defining Connectivity.....		43
Turning on Case Sensitivity		44
Customizable DFM Rule File Design		45
Grouping Rule Checks		45
Environment Variables and Pre-Processor Directives		46
Using Environment Variables in SVRF		46
Using Environment Variables in TVF		47
SVRF to TVF Comparison		48
DFM Automatic Waivers.....		50
Chapter 3		
DFM Rules and Metrics		51
Assessment Metrics		52
Writing Recommended Rule Checks		52
Saving Error Measurements as Properties		53
Coding Complex Design Rule Checks		54

Creating a Property to Reflect the Yield Impact of an Individual Error.....	57
Writing Expressions for Scores and Metrics	58
Writing Rules That Display in DFM Report Card.....	59
Writing Rule Decks For Automatic Layout Enhancement	71
Doubling Vias.....	71
Expanding Enclosures.....	75
Expanding Enclosures After Doubling Vias.....	76
DFM Rules and Metrics Verification	77
Chapter 4	
Layout Enhancement	81
Calibre YieldEnhancer Flow	83
Stage 1: Defining Inputs for Via Insertion and Enclosure Expansion	86
Environment Variables for Input Database Files	86
Rule File Modifications	86
Layermap File	89
Stage 2: Running YieldEnhancer to Add Vias and Expand Enclosures.....	91
Stage 3: Backannotating Added Vias and Expanded Enclosures.....	94
Stage 4: Defining Inputs for Adding Metal Fill	95
Stage 5: Running YieldEnhancer to Add Metal Fill.....	96
Stage 6: Backannotating Added Metal Fill.....	98
Sample Wrapper Script	99
Chapter 5	
Critical Area Analysis to Predict Yield.....	101
Calibre YieldAnalyzer and Critical Area Analysis	102
CAA Flow	103
Defect Density File Data	104
CAA Rule Deck Generation Through the CAA Tool	104
Review of Critical Area Analysis Results	104
CAA Analysis with Marker Layers	105
CAA Analysis with Exclusion Layers.....	106
CAA Analysis with Memory Redundancy	108
Running a CAA Memory Analysis.....	108
CAA Memory By-Window Analysis	109
Re-analyzing Memory Analysis Results	109
Performing Net-Based CAA	112
CAA Library Analysis.....	114
Placement Boundaries.....	114
Performing CAA What-If Analysis	115
CAA with No Defect Density	117
CAA NDD Flow.....	117
CAA NDD Metrics	118
Running a CAA NDD Analysis	119
Reviewing CAA NDD Results	120

Table of Contents

Chapter 6	
Calibre DFM Explorer	131
DFM Explorer Tool Flow	131
DFM Explorer PDK Structure	133
DFM Explorer GUI Behavior	136
Using the DFM Explorer Tool	146
Chapter 7	
Fill Layers in Calibre YieldEnhancer	157
Calibre SmartFill Flow	158
Tips for Adding Fill	159
Preventing Empty Fill Areas	160
Defining the Fill Pattern: Simple or Staggered	164
Defining Complex Polygons for Fill Shapes	165
Chapter 8	
Calibre YieldEnhancer ecofill for Fill with Design ECOs	167
ECO Fill Flow Using the ecofill Utility	167
ecofill	169
Chapter 9	
Calibre YieldEnhancer PowerVia	175
Calibre YieldEnhancer PowerVia Flow	176
Running the powervia Utility	179
Via Constraints	180
Layer Stack Definition	181
Lower Layer Via Specifications	181
Lower Layer Type (V0 case)	182
Via Addition	183
Insertion of Different Via Types (LRG, BAR, SQUARE)	183
Spacing Between Vias	185
Connectivity-Based Spacing for Vx to Vx-1 Layer Vias	187
Spacing from BAR Vias	189
Same-Layer Spacing Rule	190
Connectivity-Based Spacing for Same-Layer Vias	192
Spacing by PRL for Same-Layer Vias	193
Exclusion Areas	195
Via Count Rules Filtering	196
Original Via Layers Extraction for Extended Via Types	198
Calibre YieldEnhancer PowerVia Files and Specifications	201
Via Variables File Specifications	201
Enclosure Template File Specifications	207
Generated Enclosure Template File Specifications	208
Calibre PERC Flow File Specifications	209
Backannotation Flow Specifications	210
Net Name Specifications	213
Calibre YieldEnhancer PowerVia techlib.tcl File Specifications	214
Calibre YieldEnhancer PowerVia Command Reference	225

powervia	226
powervia Variables	230
Chapter 10	
DFM Terms and Concepts	239
Binning	240
Binning Random Particles	240
Binning Recommended Rule Violations	241
Calibre Results	241
Calibre nmDRC Results Summary	241
Calibre Transcript	243
Notification Messages for DDKs	243
Capture Window	244
Clustering	245
Concurrency for DFM Operations and Statements	245
Critical Area	247
Critical Area for Shorts	247
Critical Area for Opens	249
Defect Density	250
Derived Error Layers	251
Example Derived Error Layer	251
DFM Annotations	252
How Annotations Compare to Other Types of MetaData	252
DFM Databases	253
DFM Expressions	254
DFM Function Statement and User-Defined Functions	255
DFM Properties	256
Math Operators	257
Binary Operators	257
Unary Operators	258
Partitioning	259
Partitioning by Window	260
WINDOW Size	261
STEP Definition	261
Number of Windows	262
Design Data Boundaries	262
Partitioning by Cell	263
Pattern Substitution in DFM Commands	264
Quality Assessment Metrics	265
Results Databases	267
Calibre nmDRC Results Database	267
DFM RDB Files	269
Runsets	269
Vectors	270
Yield Assessment Metrics	270
YieldServer	271

Table of Contents

Chapter 11	
DFM Reference	273
DFM Via Shift Commands	274
Introduction to DFM Via Shift	274
DFM Spec Via Shift	275
DFM Via Shift	280
DFM Reshape Commands	298
DFM Spec Reshape	299
DFM Reshape	303
DFM RDB Reference	316
Scores RDB File Format	317
Check Results RDB File Format	321
Transitions RDB File Format	325
Single Layer RDB File Format	330
Multilayer RDB File Format	335
Fill RDB File Format	341
DFM File Formats	347
Histogram File	347
Header	347
Histogram	347
CAA Defect Data Format	350
Format (Defect Data)	351
Layer and Defect Type, Defect Range and Defect Density Data	352
Layer Names	363
CAA Layers File	366
Format	366
Custom OD Layer Names	369
Memory Configuration File Format	370
CAA Library Flow Cell Orientations	372
CAA What-if Analysis Configuration File Format	373
Fill Layermap File	374
Format (Fill Layermap)	374
DFM Analysis Reference	375
Calibre Critical Area Analysis Tool	376
Accessing the CAA Tool	376
Calibre CAA Tutorial and Example Kit	383
DFM Report Card Interface	384
Accessing the DFM Report Card Interface	384
Calibre RVE for DFM Report Card	385
Overview	385
Filtering Data by Flat Metrics	388
Tab Contents	389
Mentor Standard Layer Names	407
Appendix A	
DFM Report Card	409
DFM Report Card Controls	410
Layer Annotations	410

Table of Contents

Database-Level Annotations.....	413
Example DFM Report Card.....	419

Appendix B Foundry Certified Flows 423

Using Calibre With a UMC Rule File	424
Running Critical Area Analysis With a UMC Rule File	424
Running Calibre YieldEnhancer With a UMC Rule File	426
Using Calibre With a Common Platform Alliance Rule File	427
Running Critical Area Analysis With a Common Platform Alliance Rule File.....	427
Running Critical Feature Analysis With a Common Platform Alliance Rule File.....	428
Using Calibre With an ST Design Kit.....	429
Running Critical Feature Analysis With an ST Rule File	429
Running Calibre YieldEnhancer With an ST Rule File	429

Index

Third-Party Information

List of Figures

Figure 1-1. Basic DFM Flow.....	18
Figure 1-2. Calibre YieldServer Flow	24
Figure 2-1. Simplified Flows from Designer's Perspective	30
Figure 3-1. Test Layout	77
Figure 3-2. Comparing Layout Text to DFM Scores.....	79
Figure 4-1. Overview of Calibre YieldEnhancer Flow	85
Figure 4-2. Setting Environment Variables for Input Database Files	86
Figure 4-3. Connected Versus De-Coupled Pins.....	90
Figure 4-4. Sample Wrapper Script	99
Figure 5-1. Defect Density, Critical Area, and Lambda	102
Figure 5-2. Critical Area Analysis Flow Options Using Calibre.....	104
Figure 5-3. CAA Analysis with Marker Layers	106
Figure 5-4. Calibre Interactive DFM GUI Inputs	120
Figure 5-5. CAA NDD Totals — Score Metric Results	121
Figure 5-6. Standard CAA Totals — Yield Metric Results.....	121
Figure 5-7. CAA NDD Chip Summary — Score Metric Results	122
Figure 5-8. Standard CAA Chip Summary — Yield Metric Results	123
Figure 5-9. CAA NDD Window Summary — Score Metric Results	123
Figure 5-10. CAA NDD Window Summary Drill Down — Score Metric Results	124
Figure 5-11. CAA NDD Cell Summary — Score Metric Results.....	124
Figure 5-12. Colormap Hierarchy Windows Avg_Quality	125
Figure 5-13. Colormap Highlight Window	126
Figure 5-14. Colormap Highlight Window in Layout.....	126
Figure 5-15. Histogram Hierarchy Windows Avg_Quality.....	127
Figure 5-16. Highlight Histogram Bar.....	128
Figure 5-17. Histogram Highlight Bar in Layout	128
Figure 5-18. CAA NDD Export — Score Metric Results	129
Figure 6-1. DFM Explorer Tool Flow	132
Figure 7-1. Basic Flow for Adding Fill	158
Figure 7-2. Fill Results With NOEMPTY and TOMAX.....	162
Figure 7-3. Fill Results With Shape for Empty Regions	163
Figure 7-4. Donut Shaped Fill Shape.....	165
Figure 7-5. How to Create the Donut Shaped Fill	166
Figure 8-1. ECO Flow With the ecofill Utility	168
Figure 9-1. Calibre Yield Enhancer PowerVia Flow with the powervia Utility	177
Figure 10-1. Manufacturing Defects	247
Figure 10-2. Short Circuit Critical Area	248
Figure 10-3. Short Circuits Between Parallel Wires	248
Figure 10-4. Short Circuits Between Non-Parallel Wires	248
Figure 10-5. Open Wire Critical Area	249

Figure 10-6. Open Via Critical Area	249
Figure 10-7. RDB WINDOW and STEP	262
Figure 10-8. The Calibre YieldServer System	272
Figure 11-1. MRC Zones for Vias	278
Figure 11-2. Via Shifting	287
Figure 11-3. DFM Via Shift Optimization	291
Figure 11-4. Constrained Via Clusters	292
Figure 11-5. Balanced and Increased Coverage Within Via Clusters	292
Figure 11-6. Exclusion Shape Overlapping With Drawn Via	293
Figure 11-7. Functional Exclusion Shape Usage	293
Figure 11-8. DFM Via Shift EXTEND BY	295
Figure 11-9. DFM Via Shift Input Array With Marker Region	296
Figure 11-10. DFM Via Shift Marker Region Detail	296
Figure 11-11. DFM Via Shift Marker Region Replicated Across Array	297
Figure 11-12. DFM Spec Reshape LINE-END With DFM RESHAPE LINE-END_PULLBACK	301
Figure 11-13. DFM Spec Reshape LINE-END With DFM RESHAPE LINE-END_EXTENSION	302
Figure 11-14. Out-Corner and In-Corner Shape Transformations	308
Figure 11-15. CORNER (Input)	309
Figure 11-16. Case 1: CORNER (CORNER_CONVEX)	309
Figure 11-17. Case 2: CORNER (CORNER_CONVEX Edge Pairs)	310
Figure 11-18. Case 3: CORNER (CORNER_CONVEX and CORNER_CONCAVE with MASK)	310
Figure 11-19. Case 4: CORNER and LINE-END with MASK and RESIZE Attribute . . .	311
Figure 11-20. UNIFORM (Input)	311
Figure 11-21. Case 1: UNIFORM BY RECTANGLE (Square)	312
Figure 11-22. Case 2: UNIFORM By RECTANGLE (Square with TOLERANCE and GRID) 312	
Figure 11-23. Case 3: UNIFORM BY RECTANGLE (Rectangle with GRID)	313
Figure 11-24. Case 4: UNIFORM BY RECTANGLE (Square Resized)	313
Figure 11-25. Case 5: UNIFORM BY RECTANGLE (Rectangle Resized with TOLERANCE and GRID)	314
Figure 11-26. Case 1: DFM RESHAPE LINE-END_PULLBACK	314
Figure 11-27. Case 2: DFM RESHAPE LINE-END_EXTENSION	315
Figure 11-28. Scores RDB from DFM Analyze	318
Figure 11-29. Check Results RDB File Format	322
Figure 11-30. Transitions RDB File Format	326
Figure 11-31. Single Layer RDB File Format	331
Figure 11-32. Multilayer RDB File Format	336
Figure 11-33. Fill RDB File Format	342
Figure 11-34. Sample Histogram File	349
Figure 11-35. CAA Defect Data Format Explanation	350
Figure 11-36. Power Modeling and User-Defined List Defect Data Formats	352
Figure 11-37. Mapping of Predefined Layer Names	364

List of Figures

Figure 11-38. Mapping of User-defined Layer Names	365
Figure 11-39. Calibre Interactive DFM Library Flow Setup.....	372
Figure 11-40. Cell Orientations	373
Figure B-1. Calibre CAA with UMC Defect Data	424
Figure B-2. Calibre CAA with a Common Platform Alliance Rule File.....	427

List of Tables

Table 1-1. Syntax Conventions	26
Table 2-1. Comparing Output Database Options for DFM	39
Table 2-2. Results Generated by DFM Operations	40
Table 2-3. Comparing Results Formats	42
Table 2-4. Comparing SVRF to TVF	48
Table 3-1. Sample Spacing Ranges	52
Table 3-2. Sample Weightings for Spacing Ranges	58
Table 4-1. Supported Database Formats	81
Table 4-2. Stages of Calibre YieldEnhancer Flow	83
Table 6-1. DFME Environment Variables for Specifying Varied Calibre Versions	133
Table 7-1. Stages of Adding Fill	158
Table 7-2. Performance Based on DFM Spec Fill Shape Type	159
Table 7-3. Methods to Prevent Empty Fill Areas	161
Table 8-1. Environment Variables for ecofill Utility	173
Table 9-1. Lower-Via Spacing Keywords	188
Table 9-2. Same-Layer Spacing Keywords	191
Table 9-3. Example Enclosure Template File Lines	209
Table 9-4. Net Names Mapping	213
Table 9-5. RDB Configuration Parameters — Metal Intersection	220
Table 9-6. RDB Configuration Parameters — Original/Inserted Vias	221
Table 9-7. Variables for powervia Utility	230
Table 10-1. Supported Concurrency for DFM Operations and Statements	245
Table 10-2. DFM Operations that Support Annotations	252
Table 10-3. Summary of MetaData Stored In Databases	253
Table 10-4. Binary Operators	257
Table 10-5. Unary Operators	258
Table 10-6. Default Partitioning	259
Table 10-7. Options for Partitioning BY CELL	264
Table 10-8. Summary of Results Databases	267
Table 10-9. DFM RDB Modes	269
Table 10-10. Operations Generating Side RDBs	269
Table 11-1. DFM RDBs, Summary	316
Table 11-2. Scores RDB Result Properties for DFM Analyze	320
Table 11-3. rulecheck_name Format in Transitions RDB	326
Table 11-4. rulecheck_name Format in Single Layer RDB	332
Table 11-5. rulecheck_name Format in Multilayer RDB	337
Table 11-6. DFM Fill Statistics	345
Table 11-7. File Formats Used by DFM Operations	347
Table 11-8. Supported Defect Types Summary	358
Table 11-9. CAA-Specific Reference Topics	375

Table 11-10. Contents of Critical Area Analysis Tool	377
Table 11-11. Contents of the Totals Tab	390
Table 11-12. Contents of the Chip Summary Tab	392
Table 11-13. Contents of the Cell Summary Tab	394
Table 11-14. Contents of the Window Summary Tab	396
Table 11-15. Contents of the Drill Down Tab	398
Table 11-16. Layer Browser Filter Menu Options	402

Chapter 1

Introduction to Calibre DFM Tools

The Calibre product line includes products specifically designed to help you improve the yield of your designs.

Calibre DFM Overview	17
Calibre DFM WorkFlow	18
Key Concepts	20
Calibre YieldAnalyzer	20
Calibre YieldEnhancer	21
Calibre YieldServer	24
Calibre DFM Requirements	25
TVF Requirements	25
Syntax Conventions	26

Calibre DFM Overview

Using Calibre YieldAnalyzer and Calibre YieldEnhancer can help you prevent many types of defects that are common in any design.

The Calibre DFM product suite includes the following products:

- **Calibre YieldAnalyzer Product** — Calibre YieldAnalyzer, calculates metrics for a specific layout. This makes it possible to compare layouts, identify potential failure spots, and prioritize design modifications.
- **Calibre YieldEnhancer Product** — Calibre YieldEnhancer improves yield through automated design modifications targeted at improving certain well-characterized design weaknesses. Methods of improving yield include adding redundant vias, expanding via enclosures, and growing sensitive features.
- **Calibre YieldServer Product** — Calibre YieldServer, provides a mechanism for managing the layout design data required when designing for improved manufacturability and yield. The Calibre YieldServer provides functionality not available through the standard Calibre nmDRC hierarchical engine.

These products can even help you reduce defects that may not occur until after the chip is already in use, such as open circuits in metal layer transitions. Calibre YieldAnalyzer and Calibre YieldEnhancer are especially useful for designs below the 100 nanometer process node.

Related Topics

[Quality Assessment Metrics](#)

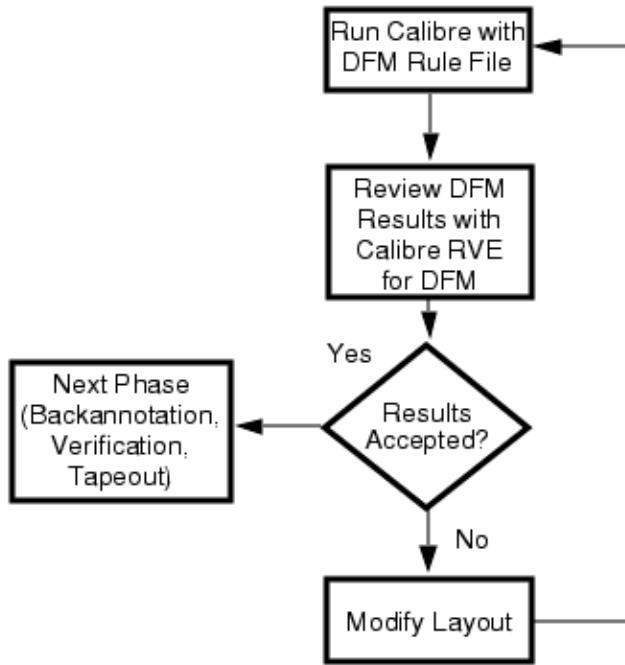
[Yield Assessment Metrics](#)

Calibre DFM WorkFlow

The Calibre DFM product has a minimum-level flow you can adapt to your design style.

[Figure 1-1](#) on page 18 shows the DFM flow, which is the basic use model for Calibre YieldAnalyzer and Calibre YieldEnhancer.

Figure 1-1. Basic DFM Flow



Both Calibre YieldAnalyzer and Calibre YieldEnhancer are based upon DFM (Design For Manufacturability) layer operations implemented through the Standard Verification Rule Format (SVRF). The flow for using the tools is similar to the Calibre nmDRC flow. It involves writing a rule file to perform checks, checking a design to find problem spots, then modifying the design to reduce the yield inhibiting effects in the problem spots and to make the design as a whole more robust.

The DFM flow differs from a Calibre nmDRC flow in a few key ways:

- **Results** — Results are reported as relative scores rather than as a passing or failing grade.
- **Rule Checking** — The rule checking performed in DFM checks your design for violations of recommended rules, not hard and fast design rules. There are always many

more recommended rule violations than design rule violations, and the designer does not need to correct each one of them. The key to successfully designing for improved yield is to correct the violations that are most detrimental to yield.

- **Critical Area Analysis** — A DFM analysis typically checks for more than rule violations. It should also check for sensitivity to random particle defects (critical area analysis). Critical area analysis helps identify those areas where recommended rule violations cause the most problems.
- **Automatic Modifications** — Modifying a design to improve yield is not limited to simply correcting rule violations. It can also include automatic modifications targeted at making the design less susceptible to failure in the field, such as improving transitions by building in redundancy.

You can incorporate the DFM flow into your existing flow by adding it as an extra process loop performed any time you run your Calibre nmDRC checks. To take fullest advantage of the flow, the rule writer should take advantage of the DFM Report Card controls.

Related Topics

[Calibre YieldServer Reference Manual](#)

[DFM Report Card](#)

Key Concepts

When incorporating DFM products into your existing flow, you must consider what potential design problems the product can address.

Calibre YieldAnalyzer	20
Calibre YieldEnhancer	21
Calibre YieldServer	24

Calibre YieldAnalyzer

Calibre YieldAnalyzer provides analysis tools for assessing yield, comparing designs, and prioritizing modifications.

Refer to the *Calibre Administrator's Guide* for complete information about licensing.

These tools include the following:

- Multi-layer spatial checks
- Run-length checks
- Width and spacing checks
- Susceptibility to random particle defects
- Via redundancy analysis and derived polygon layers

The product includes the following SVRF statements you use to create DFM layer operations tailored to your design:

- **DFM Analyze** — Provides statistical analysis of layout features for the purpose of finding issues that may affect yield. These include analysis of area, perimeter, length, and layer.
- **DFM CAF** — Calculates a numeric critical area value on one or more input layers.
- **DFM Critical Area** — Performs single-layer checks for areas in your layout that are vulnerable to defects of a given size. These defects cause open or short circuits, and can be the result of various steps in the manufacturing process.
- **DFM Function** — Creates a user-defined function for use in DFM Analyze or DFM Property expressions.
- **DFM GCA** — Outputs a derived polygon layer that represents areas susceptible to producing open or short circuits in the presence of random particles of a given radius.
- **DFM Histogram** — Analyzes the distribution of DFM Property values and computes a histogram of property values and prints out as a text file.

- **DFM Measure** — Performs single-layer, table-based width and spacing checks that help you assess the impact that certain design features have upon chip yield in the manufacturing process.
- **DFM Property** — Annotates layer data with a user-defined property value, and creates a derived layer containing all the data for which the property meets a specified criteria.
- **DFM Property Net** — Creates an empty polygon layer containing DFM properties from an input layer having connectivity information or DFM properties. This type of layer is useful when only the DFM properties are needed for the output layer.
- **DFM Redundant Vias** — Identifies redundant or non-redundant polygons on a layout and produces a derived polygon layer containing the identified polygons.
- **DFM Spec Via Redundancy** — Defines a via redundancy analysis specification that must be referenced in a DFM Redundant Vias operation.

Related Topics

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

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

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

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

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

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

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

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

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

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

[DFM Spec Via Redundancy \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

Calibre YieldEnhancer

Calibre YieldEnhancer performs automated design modification designed to enhance manufacturability and yield.

Refer to the [Calibre Administrator's Guide](#) for complete information about licensing.

The Calibre YieldEnhancer product includes the following SVRF statements you use to create DFM layer operations tailored to your design:

- **Calibre DFM SmartFill** — Generates fill layers based on fill specifications you design to fit your process and chip. This tool uses the following SVRF statements:
 - DFM Spec Fill Optimizer
 - DFM Spec Fill Shape
 - DFM Spec Fill
 - DFM Fill
- **DFM Expand Edge** — Expands edges based on DFM properties annotated onto the input layer.
- **DFM Expand Enclosure** — Generates regions of expanded enclosure for vias.
- **DFM Grow** — Expands polygons on an input layer. Polygon area, width, and spacing are taken into account during expansion.
- **DFM Joggle** — Creates an output layer from which jogs have been removed based on specified criteria.
- **DFM OR Edge** — Performs a Boolean OR operation on edges from the input layers.
- **DFM Partition** — Creates a polygon layer which represents the input layer broken into rectangles and triangles meeting specific criteria for minimum size, separation, and so forth.
- **DFM RDB** — Writes original or derived layer data to an ASCII RDB (results database).
- **DFM Reshape** — Performs geometric shape transformations on the input layer to improve via and metal enclosures for a subsequent via operation. References a DFM Spec Reshape specification.
- **DFM Shift** — Shifts polygons on the input layer based on DFM properties attached to the input layer or explicit numeric values.
- **DFM Shift Edge** — Shifts edges based on DFM properties attached to the input layer.
- **DFM Size** — Oversizes or undersizes the polygons on the input layer based on values annotated onto it.
- **DFM Spec Reshape** — Defines a specification with a set of correction values used by a DFM Reshape operation to transform corner and line-end metal shapes for improved via enclosure.
- **DFM Spec Via** — Defines a via specification that can be used by DFM Transition USEVIAS.

- **DFM Spec Via Shift** — Specifies via shifting values and constraints to improve via location and metal coverage, reducing MRC violations. Called by DFM Via Shift.
- **DFM Transform** — Transforms geometries on a layer.
- **DFM Transition** — Places extra vias and metal at interconnect junctions (transitions) in locations where extra vias are permitted by the design. This improves the long-term reliability of these transitions.
- **DFM Via Shift** — Improves via location and metal coverage, reducing MRC violations. Used with DFM Spec Via Shift.

Related Topics

[DFM Expand Edge \[Standard Verification Rule Format \(SVRF\) Manual\]](#)
[DFM Size \[Standard Verification Rule Format \(SVRF\) Manual\]](#)
[DFM Expand Enclosure \[Standard Verification Rule Format \(SVRF\) Manual\]](#)
[DFM Jogclean \[Standard Verification Rule Format \(SVRF\) Manual\]](#)
[DFM OR Edge \[Standard Verification Rule Format \(SVRF\) Manual\]](#)
[DFM Partition \[Standard Verification Rule Format \(SVRF\) Manual\]](#)
[DFM Spec Fill \[Standard Verification Rule Format \(SVRF\) Manual\]](#)
[DFM Spec Fill Optimizer \[Standard Verification Rule Format \(SVRF\) Manual\]](#)
[DFM Spec Fill Shape \[Standard Verification Rule Format \(SVRF\) Manual\]](#)
[DFM Fill \[Standard Verification Rule Format \(SVRF\) Manual\]](#)
[DFM Reshape](#)
[DFM Spec Reshape](#)
[DFM Spec Via \[Standard Verification Rule Format \(SVRF\) Manual\]](#)
[DFM Spec Via Shift](#)
[DFM Shift \[Standard Verification Rule Format \(SVRF\) Manual\]](#)
[DFM Shift Edge \[Standard Verification Rule Format \(SVRF\) Manual\]](#)
[DFM Transition \[Standard Verification Rule Format \(SVRF\) Manual\]](#)
[DFM Via Shift](#)
[DFM Grow \[Standard Verification Rule Format \(SVRF\) Manual\]](#)
[DFM Transform \[Standard Verification Rule Format \(SVRF\) Manual\]](#)
[DFM RDB \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

Calibre YieldServer

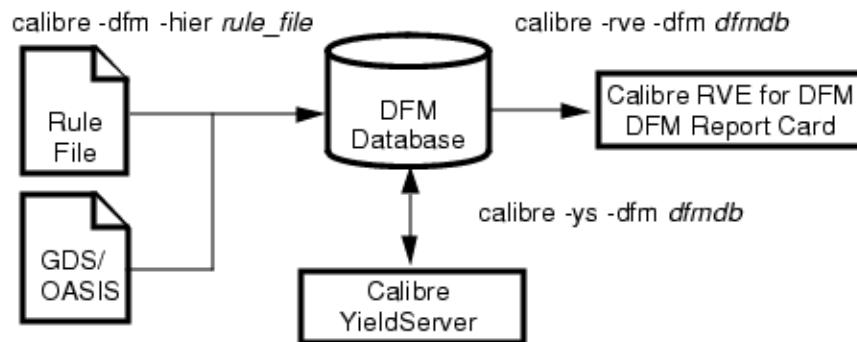
Calibre YieldServer is a software system design specifically for managing the layout design data required with designing for improved manufacturability and yield.

Refer to the [Calibre Administrator's Guide](#) for complete information about licensing.

There are four components to Calibre YieldServer:

- **DFM Database** — A layout database stored in the Siemens proprietary DFM database format, which is designed to store hierarchical layout objects with associated attributes and properties.
- **Calibre YieldServer Engine** — Data processing software that lets you query, manipulate, modify, and create data in a DFM database.
- **Calibre YieldServer Command Language** — A Tcl-based language for interfacing with the Calibre YieldServer engine.
- **DFM Report Card** — A set of commands and controls built into the Calibre Results Viewing Environment (RVE) for DFM to provide a graphical user interface to the DFM database.

Figure 1-2. Calibre YieldServer Flow



You create a DFM database from a standard layout database using the calibre -dfm executive. Two special purpose SVRF statements control how this database is created:

- **DFM Database** — Used only with calibre -dfm, this statement defines the contents and locations of a DFM database to be created.
- **DFM Select Check** — Used only with calibre -dfm, this statement defines the rule checks to evaluate.

Related Topics

[Calibre YieldServer Reference Manual](#)

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

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

DFM Report Card

Calibre DFM Requirements

Before running Calibre YieldAnalyzer or Calibre YieldEnhancer, you must have a full Calibre installation and the appropriate licenses.

The following licenses required to run these tools are as follows:

- Calibre YieldAnalyzer license
- Calibre YieldEnhancer license

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

In general, the inputs to Calibre DFM are a layout database and a SVRF rule file.

- **One or More Layout Databases** — GDS or OASIS^{®1} databases containing the design layer or layers that you need to analyze or enhance with DFM.
- **One or More Rule Files** — ASCII files composed of legal Standard Verification Rule Format (SVRF) or Tcl Verification Format (TVF) statements. These statements control:
 - The design environment.
 - Loading of data from the GDS or OASIS database.
 - Layer preprocessing such as sizing, creating or isolating new layers, and so on.
 - Design and manufacturability checks.
 - Writing the final results to one or more results databases.

Calibre tools require that the CALIBRE_HOME environment variable be set.

Related Topics

[Calibre Administrators Guide](#)

[Setting the CALIBRE_HOME Environment Variable \[Calibre Administrator's Guide\]](#)

TVF Requirements

If you intend to create DFM scripts using TVF, then there are certain requirements you must meet.

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.

Specifically, you must add the lappend and package require statements to provide access to the DFM script package. For example:

```
lappend auto_path [file join $env(CALIBRE_HOME) pkgs icv tvf dfm]
package require CalibreDFM_YLD
```

Related Topics

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

Syntax Conventions

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

Table 1-1. Syntax Conventions

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

Table 1-1. Syntax Conventions (cont.)

Convention	Description
Example: DE vice { <i>element_name</i> [‘(‘ <i>model_name</i> ‘)’]} <i>device_layer</i> { <i>pin_layer</i> [‘(‘ <i>pin_name</i> ‘)’] ...} [‘<’ <i>auxiliary_layer</i> ‘> ...] [‘(‘ <i>swap_list</i> ‘)’ ...] [BY NET BY SHAPE]	

Chapter 2

Getting Started with Calibre DFM

The move to Design for Manufacturability (DFM) is about developing closer associations between the design world and the manufacturing world.

The goal is to help layout design engineers choose design strategies that result in more manufacturable (higher yielding) layout configurations. From the designer's perspective, it is important that the move to DFM does not require that they learn a whole new way of designing and checking their work.

Calibre YieldAnalyzer and YieldEnhancer provide a set of Standard Verification Rule Format (SVRF) statements and operations you can use to develop DFM rule files that use the same tools and processes your designers are using for standard DRC.

Calibre DFM Command Line Invocation	30
Calibre Command Line Modes.....	30
calibre	32
Invocation of Calibre DFM from Calibre Interactive	36
DFM Design Environment Management.....	36
DFM Rule File Basics	37
Calibre DFM Output Database Format	39
Saving DFM Results to a DFM Database.....	39
Saving DFM Results to an RDB.....	40
DFM-Specific SVRF Rule File Entries	43
Defining Connectivity.....	43
Turning on Case Sensitivity	44
Customizable DFM Rule File Design.....	45
Grouping Rule Checks	45
Environment Variables and Pre-Processor Directives	46
Using Environment Variables in SVRF	46
Using Environment Variables in TVF	47
SVRF to TVF Comparison	48
DFM Automatic Waivers	50

Calibre DFM Command Line Invocation

The Calibre YieldAnalyzer and Calibre YieldEnhancer software is executed by the Calibre engine, either flat or hierarchical, that you invoke from the command line. Running Calibre DFM from the command line is the preferred method for full chip analysis or enhancement when the layout database is supplied in OASIS or GDS format.

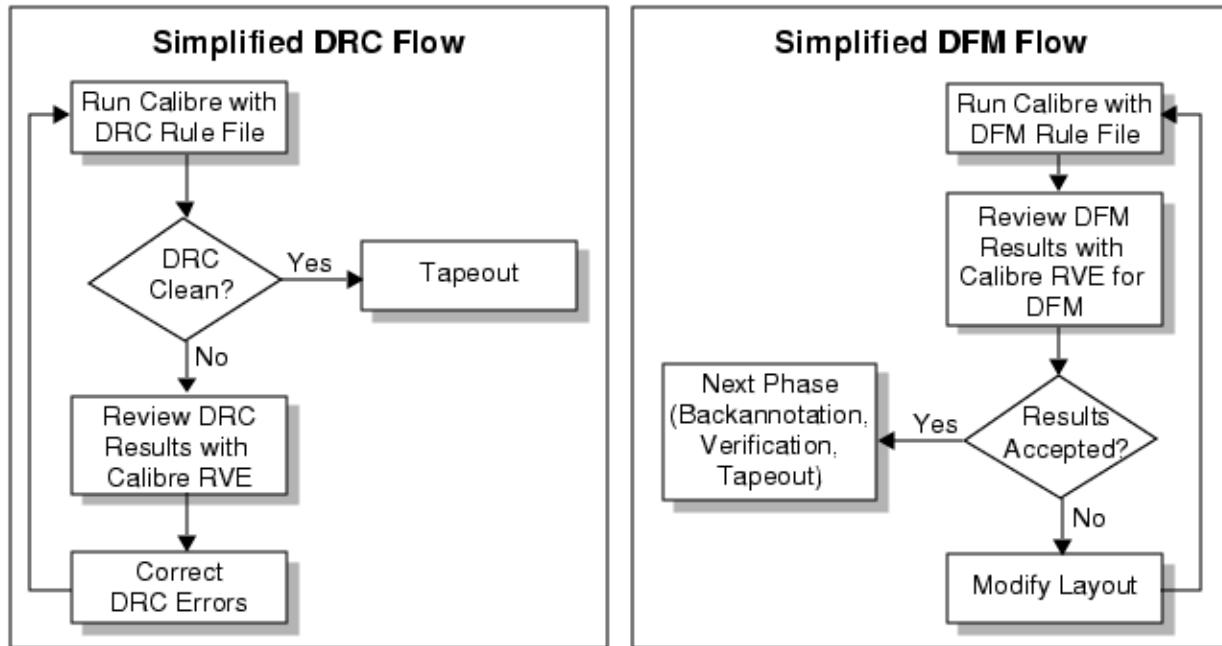
Calibre Command Line Modes	30
calibre	32

Calibre Command Line Modes

The command line invocation mode, that is using either the -drc or -dfm switch, varies depending on the way the rule file has been constructed, and your needs with respect to results interpretation.

Figure 2-1 on page 30 contrasts the flow of the DRC and DFM invocation modes.

Figure 2-1. Simplified Flows from Designer's Perspective



Referring to Figure 2-1 on page 30, the primary differences between the two processing modes is as follows:

- **-drc** — Results are written to either an ASCII results database or a layout database, as specified through the DRC Results Database statement.
- **-dfm** — Results are written to a DFM database, as specified through the DFM Database statement.

Related Topics

[calibre](#)

[Invocation of Calibre DFM from Calibre Interactive](#)

calibre

This command is used to invoke the Calibre hierarchical engine.

Usage

```
calibre { -drc -hier | -dfm -hier }
[ { { -turbo [ number_of_processors ] }
  | { -turbo_litho [ number_of_processors ] }
  | { -turbo [ number_of_processors ] -turbo_litho [ number_of_processors ] } }
  [ -turbo_all ]
  [ { -remote host,host, ...
    | -remotefile filename
    | -remotecommand filename count }
    [ -remotedata [ -recoverremote | -recoveroff ] ] ]
  [ -hyper [remote]]
]
[ -nowait | -wait n ]
[ -lmconfig licensing_config_filename ]
[ -E svrf_output_from_tvf ] [ -tvfarg tvf_argument ]
[ -validprop [report_options | all] [trace]]
rule_file_name
```

Arguments

- **-drc -hier | -dfm -hier**

A required argument set that selects the processing mode to use. One and only one of these arguments must be specified.

- **-drc -hier** — Runs hierarchical Calibre in Calibre nmDRC mode. Results are written to the results database(s) specified by the DRC Results Database statement.
- **-dfm -hier** — Runs hierarchical Calibre in DFM mode. Results are written to the DFM database specified by the DFM Database statement. You must specify the **-hier** option if you specify **-dfm**.

- **-turbo [number_of_processors]**

An optional argument that enables multithreaded parallel processing.

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

Calibre runs on the maximum number of CPUs available if you specify a number greater than the maximum available.

- **-turbo_all**

An optional argument that halts Calibre execution if the tool cannot obtain the exact number of CPUs you specified using -turbo or -turbo_litho, or both. The -turbo_all option is used with the -turbo and -turbo_litho options.

For example:

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

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

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

Without -turbo_all, the Calibre tool normally uses fewer threads than requested if the requested number of licenses or CPUs is unavailable. The -turbo_all argument is ignored if loop licensing is used.

- **-turbo_litho [number_of_processors]**

An optional argument that enables multithreaded parallel processing for RET and MDP operations.

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

You can specify both the -turbo and the -turbo_litho options and the respective *number_of_processors* arguments can vary between the two options.

- **-remote host[, host ...]**

An optional argument that specifies to use the Calibre® MTflex™ parallel processing architecture for running multithreaded Calibre tools on distributed networked computers. It can only be specified with the -turbo or -turbo_litho argument. The network must be homogeneous when -remote is specified.

You must specify at least one host parameter. To run on multiple hosts, specify a comma-separated list of hosts. The hosts must all have the same platform type. You must have the required number of licenses for your job.

- **-remotefile filename**

An optional argument that specifies to use the Calibre MTFlex multithreaded parallel processing architecture. It can only be specified with the -turbo or -turbo_litho argument. The filename specifies the pathname of a configuration file containing information for the local and remote hosts. You must have the required number of licenses for your job.

The remote hosts can be heterogeneous (have different platform types) when using this option.

See “[-remotefile](#)” in the *Calibre Administrator’s Guide* for more information.

- **-remotecmd *filename count***

An optional argument set that causes a Calibre MTflex configuration file to be automatically generated using the supplied parameters. This option can only be used with the -turbo or -turbo_litho argument. It may not be specified with -remotefile or -remote.

- *filename* — The path to an executable file on the local host containing the commands for invoking a remote server on the remote host. The *filename* is used as the first argument in the REMOTE COMMAND statement in the generated configuration file.
- *count* — The total number of CPUs that Calibre attempts to use for the run. The *count* value must be greater than or equal to 2. This value is used for the COUNT parameter in the LAUNCH CLUSTER statement in the generated configuration file.

This option is useful for Calibre MTflex runs in the IBM® Platform™ LSF® environment because it eliminates the need for a configuration file.

See “[-remotecmd](#)” in the *Calibre Administrator’s Guide* for more information.

- **-remotedata [-recoverremote | -recoveroff]**

An optional argument set that enables the Calibre Remote Data Server (RDS) technology. This technology is designed to reduce local (server) host memory requirements and improve Calibre MTflex performance by distributing data across remote hosts. This argument set can only be used when -remote, -remotefile, or -remotecmd is specified.

See “[-remotedata](#)” in the *Calibre Administrator’s Guide* for details.

- **-hyper [remote]**

An optional argument set that enables hyperscaling mode, which enables the concurrent, parallel execution of SVRF operations, thereby increasing the utilization of CPUs and improving scalability in the MT and Calibre MTflex environments. This option can only be specified with -turbo or -turbo_litho.

The optional remote argument triggers remote pseudo-hierarchical database technology and enables the Remote Data Server (RDS) technology (the -remotedata argument) by default. The remote argument can only be used with the -remote, -remotefile, or -remotecmd arguments.

See “[Calibre Command Line](#)” in the *Calibre Administrator’s Guide* for details on the -hyper argument, including remote and local host requirements.

- **-nowait**

An optional argument that causes Calibre to queue only briefly (approximately 10 seconds) before attempting to acquire substitute licenses. This argument is equivalent to specifying “-wait 0”.

- **-wait *n***

An optional argument that specifies the maximum time in minutes for Calibre to queue for a specific license. If the license is unavailable after queueing for *n* minutes, Calibre attempts

to acquire any substitute licenses or exits if no suitable substitutions are defined. The maximum value for n is 45000, which is slightly more than one month.

See “[Calibre Licensing Command Line Options](#)” in the *Calibre Administrator’s Guide* for details.

- `-lmconfig licensing_config_filename`

An optional argument that specifies the path to a license configuration file that specifies the licensing mode and parameters.

See “[Calibre Licensing Configuration File](#)” in the *Calibre Administrator’s Guide*.

- `-E svrf_output_from_tvf`

An optional argument set that may be used when the rule file is in compile-time TVF format. This argument causes the TVF pre-processor to save the generated SVRF rule file to the file `svrf_output_from_tvf`. By default, only the TVF rule file is echoed to the transcript.

If the `-E` argument set is specified with a rule file but no other arguments, Calibre runs in preprocessor only mode to generate the SVRF code and write it to the specified file. A Calibre run is not started.

See “[Tcl Verification Format](#)” in the *Standard Verification Rule Format (SVRF) Manual*.

- `-tvfarg tvf_argument`

An optional argument set that may be used when the rule file is in compile-time TVF format. The string `tvf_argument` is passed to the compile-time TVF script. The `tvf_argument` must not contain space characters. The `tvf_argument` is read by the `tvf::get_tvf_arg` command in the TVF rule file.

- `-validprop [{ [missing] [unused] [outputonly] } | all] [trace]`

An optional argument set that invokes the property validator, which checks a rule file for missing and unused DFM properties. You must also specify `-drc` or `-dfm`, and other arguments necessary for a Calibre run with the specified rule file.

report_options — Specify one or more of the following to control what properties are reported on.

- `missing` — Report properties that are used but not defined.
- `unused` — Report properties that are not used.
- `outputonly` — Report properties that are used only in output.

For example, properties are included in output from a DFM RDB operation or a DRC Check Map statement with the PROPERTIES keyword.

- `automatic` — Report properties that are always generated by an operation, regardless of any user-specified options. May be abbreviated as `auto`.

`all` — Report missing, unused, outputonly, and automatic properties. This is the default.

`trace` — Include a trace of the property analysis in the output.

The property validator examines a limited set of rule file operations; see “[Property Validator](#)” in the *Standard Verification Rule Format (SVRF) Manual*.

- ***rule_file_name***

A required argument that specifies the pathname of the rule file.

Related Topics

[Calibre Licensing Configuration File \[Calibre Administrator's Guide\]](#)

[Licensing for Multithreaded Operations \[Calibre Administrator's Guide\]](#)

[Command Line Options Reference Dictionary \[Calibre Administrator's Guide\]](#)

[Using License Substitution With Calibre Classic Licensing \[Calibre Administrator's Guide\]](#)

Invocation of Calibre DFM from Calibre Interactive

Invoking Calibre DFM from Calibre Interactive is the preferred method for partial chip analysis or enhancement, or when the layout database is supplied in a format other than OASIS or GDS.

This is because Calibre Interactive DFM lets you launch DFM runs that perform the DFM checks or operations on any layout you have open in your layout editor. This allows you to check a design modification before you commit to it.

Related Topics

[calibre](#)

[Using Calibre Interactive to Perform DFM \[Calibre Interactive User's Manual\]](#)

DFM Design Environment Management

The power of DFM comes largely from the ability to compare and contrast designs with respect to manufacturability concerns.

This kind of comparison typically involves many files that you need to manage. You can expect to work with the following types of files:

- Rule files
- Layer files
- Runset files for Calibre Interactive DFM
- Results databases
- Original layout databases

- Modified layout databases

For best results, develop a plan for managing your DFM design environment before you begin analyzing and modifying your designs. Consider the following when developing your plan:

- **Naming Conventions** — Design a naming convention to clearly distinguish between types of files and the designs with which they are associated, such as:

design_name'_'type.extension

where:

type — scores, results, runset, rules, and so on.

extension — *.tvf*, *.svrf*, *.txt*, *.gds*, and so on.

Additionally, it is good practice to specify a different run directory for each design or analysis.

- **Calibre Interactive DFM Runsets** — Create one Calibre Interactive runset for each type of analysis or modification you expect to perform. Use the controls available through Calibre Interactive DFM to target each runset to the exact type of analysis or enhancement you want to perform.

For example, you can specify a custom Check Selection Recipe which selects the rule checks to execute for the run.

- **Unique Pathnames** — Reduce the risk of overwriting data files by defining a unique “seed” pathname for results, summaries, and other files, such as

design_name/run_type/analysis_I

You can increment the ordinal for the output filename every time you perform the analysis.

Related Topics

[About Calibre Interactive Runsets \[Calibre Interactive User's Manual\]](#)

[Check Selection Recipe \[Calibre Interactive User's Manual\]](#)

DFM Rule File Basics

The Calibre DFM products require an SVRF rule file as input similar to Calibre nmDRC. There are, however, differences between a DFM and a nmDRC SVRF rule file.

A DFM rule file may differ from Calibre nmDRC rule files in the following ways:

- **Inputs** — Many of the DFM operations retain connectivity data, if available. If you intend to backannotate design modifications into DEF, follow the steps described in

“[Defining Connectivity](#)” on page 43 for all input layers for which connectivity must be maintained, whether original or derived.

- **Invocation** — The SVRF operations included in Calibre YieldAnalyzer and Calibre YieldEnhancer can be evaluated using either the calibre -drc invocation or the calibre -dfm invocation. The decision as to which method to use must be made based on the type of outputs required.
- **Outputs** — Results from DFM rule checks can be output in a variety of formats. The most commonly used formats are as follows:
 - ASCII Results Databases
 - DFM Databases

While DFM rule files can be written completely in SVRF, they are extremely well suited to using TVF.

Related Topics

[Calibre DFM Command Line Invocation](#)

[DFM RDB Files](#)

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

Calibre DFM Output Database Format

The Calibre DFM products can output a Results Database (RDB) and a DFM database.

The following table summarizes the differences between the DFM output formats.

Table 2-1. Comparing Output Database Options for DFM

Format	Pros	Cons
DFM RDB File	<ul style="list-style-type: none"> Designers who are familiar with the Calibre DRC flow are used to using this format. Stores object properties if they are created using DFM Property. 	<ul style="list-style-type: none"> Larger file sizes. Does not support storage of object attributes.
DFM Databases	<ul style="list-style-type: none"> Smallest file size. Stores object attributes. Stores object properties. Supports interactive drill-down. 	<ul style="list-style-type: none"> Proprietary.

Saving DFM Results to a DFM Database	39
Saving DFM Results to an RDB	40

Saving DFM Results to a DFM Database

You generate a DFM database by running `calibre` in DFM mode, which requires that you invoke the tool using the `-dfm` switch.

Invoking the Calibre Hierarchical engine this way causes it to write its results in the form of a DFM database, rather than a Calibre nmDRC results database or an RDB.

Procedure

1. Use the rule file used as input to a `-dfm` run that is similar to a rule file used with the `-drc` switch, with the following exceptions:
 - The rule file must contain a DFM Database specification statement, which defines the contents and location of the database to be created.
 - Device recognition is supported. Note that there are some constraints on connectivity extraction and device recognition using `-dfm`.

- You cannot generate any of the DFM RDBs when using -dfm. Instead, all database output is directed to the DFM Database.
2. Invoke the Calibre Hierarchical engine with the rule file.

Related Topics

[Connectivity Extraction and Device Recognition Using -dfm \[Calibre YieldServer Reference Manual\]](#)

[DFM Databases](#)

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

[DFM RDB Files](#)

[Calibre DFM Command Line Invocation](#)

Saving DFM Results to an RDB

Many of the DFM operations produce results containing types of data that the standard Calibre nmDRC Results Database does not handle.

When this is the case, the operations can be configured to generate one or more DFM RDB Files. Keywords and arguments give you control over the pathnames for the DFM RDB output databases, as well as how data in the database is structured.

Using the correct RDB format ensures that you preserve all the information extracted from or added to the design.

Procedure

1. Study the following table for instructions on which type of database to generate for different types of data related to a Critical Area Analysis run.

Table 2-2. Results Generated by DFM Operations

Operation	Results Type	Save as...	Instructions
DFM Analyze	Property values	Scores RBD	Include the RDB keyword in the operation statement.
DFM Critical Area	Polygons	Calibre nmDRC Results Database	DRC Check Map If using TVF and the DFM TVF package, use the set_calDFM_var command to create a variable with the name “CAA_DB_layer”.

Table 2-2. Results Generated by DFM Operations (cont.)

Operation	Results Type	Save as...	Instructions
DFM Expand Enclosure	Polygons	Calibre nmDRC Results Database	DRC Check Map Or write the results layers to a DFM RDB Files using the DFM RDB operation with the NODAL keyword.
DFM Fill	Polygons	Calibre nmDRC Results Database	DRC Check Map Or write the results layers to a DFM RDB Files using the DFM RDB operation with the NODAL keyword.
DFM Fill	Statistics	Fill RDB	Include RDB keyword in the fill specification you create using DFM Spec Fill .
DFM Grow	Polygons	Calibre nmDRC Results Database	DRC Check Map Or write the results layers to a DFM RDB Files using the DFM RDB operation.
DFM Measure	Edge clusters	Check Results RDB	Include RDB keyword in the rule check. This saves the check results to a DFM RDB Files .
Recommended rules	Errors	Calibre nmDRC Results Database	DRC Check Map Or write the results layers to a DFM RDB Files using the DFM RDB operation.
DFM Transition	Polygons	Transitions RDB	Include RDB keyword in the rule check Or explicitly write the results layers to a DFM RDB Files using the DFM RDB operation with the TRANSITION and NODAL keywords.

- For results to be written to a Calibre nmDRC Results database, use DRC Check Map to control how and where the results are written. By default, this operation writes the results to the database specified in the DRC Results Database Statement. You can use

arguments to DRC Check Map To Write to other databases, which can be any of three formats: OASIS, GDS, ASCII or BINARY.

Table 2-3. Comparing Results Formats

Type	Size	Comment
OASIS	smallest	Can be loaded into Calibre® DESIGNrev™ or other viewer
GDS		Can be loaded into Calibre DESIGNrev or other viewer
ASCII BINARY	largest	Can be loaded into Calibre® RVE™ for DFM

One key consideration is “How is this data used?”

- If the goal is to support viewing the errors within the original database, ASCII is recommended because you can use it with Calibre RVE for DFM to highlight errors in layout databases of any format.
 - If the goal is to save data for further analysis or processing, OASIS is recommended because files are smallest and fast to load.
3. For results to be written to one of the DFM RDB formats, you must include the RDB keyword in the DFM operation. Arguments to this keyword control the pathname for the RDB created and the partitioning used to organize the data.
 4. For modified versions of results from any of the Calibre YieldEnhancer tools, save results to a derived layer, perform post processing on results, then use the DFM RDB operation to create the RDB containing the post-processed derived layers.

Related Topics

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

[DRC CHECK MAP \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

[DFM RDB Files](#)

[DRC RESULTS DATABASE \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

DFM-Specific SVRF Rule File Entries

The DFM SVRF rule file you input to the Calibre DFM products must define connectivity and allow for case sensitivity. Follow the additional steps required to include this information in your rule file.

Defining Connectivity	43
Turning on Case Sensitivity	44

Defining Connectivity

One of the primary differences between Calibre nmDRC checks and DFM checks is that some of the DFM operations are node-aware.

The node-aware DFM Operations receive connectivity information through compile-time checks for node-preserving derivation from Connect operations.

Prerequisites

- Either the layout database is provided in annotated GDS format, or you have connectivity data available.

Procedure

1. Include commands to gain access to or define connectivity data:
 - Working with an annotated GDS, in which each polygon has a net name property associated with it, extract the net name as text:

```
DRC CELL TEXT YES
LAYOUT PROPERTY TEXT 126
TEXT LAYER 40
ATTACH 40 metall1
```

Be sure to supply the correct values for Text Layer and Layout Property Text and repeat as needed.

- Working with a non-annotated GDS file, define a text layer to manage net properties, then use Attach to associate the text layer to the design layer:

```
TEXT LAYER 40
ATTACH 40 metall1
```

2. Define connections using the Connect or Sconnect statements:

```
CONNECT M1 DIFF BY CONT
CONNECT M1 POLY1 BY CONT
CONNECT M2 M1 BY VIA1
CONNECT M3 M2 BY VIA2
CONNECT M4 M3 BY VIA3
```

Related Topics

- [Attach \[Standard Verification Rule Format \(SVRF\) Manual\]](#)
- [Sconnect \[Standard Verification Rule Format \(SVRF\) Manual\]](#)
- [Connect \[Standard Verification Rule Format \(SVRF\) Manual\]](#)
- [Text Layer \[Standard Verification Rule Format \(SVRF\) Manual\]](#)
- [Layout Property Text \[Standard Verification Rule Format \(SVRF\) Manual\]](#)

Turning on Case Sensitivity

When writing rule files used to enhance the manufacturability of a design, you must consider the special needs of backannotation to DEF.

In particular, check to see whether or not net names are case sensitive.

Prerequisites

- You know you must backannotate to DEF.
- You believe net names may be case sensitive.

Procedure

1. If net names are or may be case sensitive, add the following line to your rule file:

```
LAYOUT PRESERVE CASE YES
```

2. Take care to use the proper case treatment for all layers and nets referenced in the rule file.

Customizable DFM Rule File Design

Calibre Interactive DFM provides several methods for customizing a Calibre rule file. Some methods, like selecting checks or defining inputs and outputs for the runs, do not involve any special coding in the rule file.

Other methods let you extend your control over a run by using coding techniques within your rule file.

Grouping Rule Checks	45
Environment Variables and Pre-Processor Directives.....	46
Using Environment Variables in SVRF.....	46
Using Environment Variables in TVF	47

Grouping Rule Checks

When your rule file contains multiple critical area and recommended rule checks, you can group related checks under a single name.

Groups simplify the task of selecting checks through Calibre Interactive DFM.

Procedure

1. Design groupings to reflect the relationships between checks that make the most sense in your design or process. Suggestions for groupings include the following:
 - By layer.
 - By check type (width, spacing, overlap, and so on).
 - By check style (critical area versus recommended rules).
2. Add one Group statement for each group you want to create using the following as a guide:

```
GROUP POLY      POLY.?
GROUP Metal1   M1.?
GROUP LIL       LIL.?
GROUP PPLUS     PPLUS.?
GROUP NPLUS     NPLUS.?
GROUP PEXT      PEXT.?
GROUP NEXT      NEXT.?
GROUP SIPROT    SIPROT.?
GROUP METAL_VIA METAL? VIA?
```

There are both SVRF and TVF versions of the Group command. Both support the “?” wildcard for simplifying group assignments.

Related Topics

Group [Standard Verification Rule Format (SVRF) Manual]

Environment Variables and Pre-Processor Directives

By using variables in your rule file, a single rule file can perform multiple functions.

This is extremely useful in DFM rule files in which you may want to vary the checks you run, the files you write to, or the way you partition the data. You can use either of the following methods to define variables:

- **SVRF** — See “[Using Environment Variables in SVRF](#)” on page 46.
- **TVF** — See “[Using Environment Variables in TVF](#)” on page 47.

For added flexibility, you can use environment variables instead of local variables, and take advantage of the Calibre Interactive DFM environment variable controls. This provides you with the ability to control exactly which processes are run through the Calibre Interactive DFM user interface.

Using Environment Variables in SVRF

SVRF provides only limited support for variables, and there are many restrictions on their use.

Prerequisites

- SVRF rule file

Procedure

1. In the SVRF rule file, use #DEFINE statements to access environment variables that define paths to files. For example:

```
#DEFINE $DFM_RDB_NAME
```

Note

 Calibre Interactive DFM recognizes that these are environment variables by the dollar sign character (\$) before the variable name and the presence of #DEFINE or #IFDEF.

2. In the SVRF rule file, use #IFDEF statements to create conditional structures controlling which operation to perform. For example:

```

#define $BY_CELL
    Gate_scores = DFM ANALYZE Gate_Space_1
    Gate_Space_2 Gate_Space_3
    Gate_Space_4 Gate_Space_5 >=0
    [ ((COUNT(Gate_Space_1))*0.000125
    + (COUNT(Gate_Space_2))*8e-05
    + (COUNT(Gate_Space_3))*4.5e-05
    + (COUNT(Gate_Space_4))*2e-05
    + (COUNT(Gate_Space_5))*5e-06)]
    BY CELL
    RDB $DFM_RDB_NAME
#else
    Gate_scores = DFM ANALYZE Gate_Space_1
    Gate_Space_2 Gate_Space_3
    Gate_Space_4 Gate_Space_5 >=0
    [ ((COUNT(Gate_Space_1))*0.000125
    + (COUNT(Gate_Space_2))*8e-05
    + (COUNT(Gate_Space_3))*4.5e-05
    + (COUNT(Gate_Space_4))*2e-05
    + (COUNT(Gate_Space_5))*5e-06)]
    WINDOW
    RDB $DFM_RDB_NAME
#endif

```

For more information on these conditional structures, refer to “[Pre-Processor Directives](#)” in the *Standard Verification Rule Format (SVRF) Manual*.

Using Environment Variables in TVF

TVF provides more robust support for environment variables, allowing you to build SVRF calls out of strings.

When developing rule files in TVF, you can use variables for the following:

- Data partition definitions for analysis.
- Values for recommended rules.
- Particle sizes for critical area analysis.
- Pathnames to output files.

Prerequisites

- A TVF file—Refer to “[Tcl Verification Format](#)” in the *Standard Verification Rule Format (SVRF) Manual*.

Procedure

1. Figure out how you build SVRF operations out of literal strings and string variables.

2. Use “dummy” #IFDEF statements and the dollar sign character (\$) before the variable name to let Calibre Interactive DFM recognize that these are environment variables.

```
// dummy IFDEF for Calibre Interactive
#ifndef $Partition_model
#endif
#ifndef $DFM_RDB_NAME
#endif
```

Note that these statements are not used by TVF or Tcl.

3. Create the Tcl variables you need, setting the values from the environment variables. Be sure to create default values in case the environment variables are not defined:

```
# see if following global variables defined in shell
if { [info exists env(DFM_RDB_NAME) ] == 1 } {
    set DFM_RDB_NAME $env(DFM_RDB_NAME);
} else
    set DFM_RDB_NAME "scores.rdb";
}
if { [info exists env(Partition_model) ] == 1 } {
    set Partition_model $env(Partition_model);
} else
    set Partition_model "BY CELL NOPSEUDO";
}
```

4. Write TVF code to build SVRF operations out of literal strings and string variables.

```
SETLAYER Gate_scores = "DFM ANALYZE Gate_Space_1
Gate_Space_2 Gate_Space_3
Gate_Space_4 Gate_Space_5 >=0
[ ((COUNT(Gate_Space_1))*0.000125
+ (COUNT(Gate_Space_2))*8e-05
+ (COUNT(Gate_Space_3))*4.5e-05
+ (COUNT(Gate_Space_4))*2e-05
+ (COUNT(Gate_Space_5))*5e-06)]
$Partition_Model
RDB \"$DFM_RDB_NAME\""
```

SVRF to TVF Comparison

Whether to use SVRF or TVF for the DFM rules depends on a number of issues.

[Table 2-4](#) provides a comparison between the two formats.

Table 2-4. Comparing SVRF to TVF

	Pure SVRF	TVF
Read parameter inputs from a file	No. Exception: setup or config files, such as LITHO setup files.	Yes.

Table 2-4. Comparing SVRF to TVF (cont.)

	Pure SVRF	TVF
Support for macros and procedures	Limited.	Yes.
Conditional statements (IF...THEN...ELSE), and loops (WHILE)	Limited, using IFDEFs plus global or environment variables.	Yes.
Access Calibre layer data during the run	No.	Yes: runtime TVF only.
Support for rule file variables	Limited, using VARIABLE statement. Serious limitation: layer names cannot be variables.	Yes.
File size	May be large.	Can be significantly smaller, mostly due to conditional statements (IF...THEN...ELSE), and loops (WHILE).
Required background knowledge	SVRF.	SVRF, Tcl.
Debugging	Must debug each line.	Must debug Tcl, then debug SVRF generated by Tcl. Compile errors and warnings help with debugging Tcl. For SVRF, you typically only need to debug the first instance of an operation, then fix the TVF that generated it.
Readability	Can be challenging for long, complex rule files.	Generally more readable because of order dependence and the fact that compile-time TVF rule files that are “condensed” SVRF.
Modularity	Supported through rulecheck GROUPs and use of INCLUDE files.	Fully supported using GROUP, INCLUDE files, re-usable procedures and so on.

Table 2-4. Comparing SVRF to TVF (cont.)

	Pure SVRF	TVF
Maintenance	Can be challenging. For example, including new layers may require duplicating large amounts of code and migrating to a new process node can require modifying each instance of a layer name, rule, or other control.	Simplified through proper use of foreach, while, if, and variables. Extending the deck to include new layers can be as simple as adding a few new elements to an array.
Version Control	Must be implemented though external version control systems.	Some version control built into Tcl. Requires placing each module inside its own package.

DFM Automatic Waivers

DFM automatic waivers supports Critical Feature Analysis (CFA).

DFM supports CFA through rules that use [DFM Analyze](#) and [DFM Property](#). This flow is similar to DRC and ERC automatic waiver, except that it produces a binary DFM Database rather than ASCII results databases.

For more details about how to set up an automatic waiver run, see “[Specifying Automatic Waivers for DFM CFA](#)” in the *Calibre Interactive User’s Manual*.

Related Topics

[Requirements for Running Calibre Auto-Waivers \[Calibre Auto-Waivers User’s and Reference Manual\]](#)

Chapter 3

DFM Rules and Metrics

Some key rule writing techniques for developing DFM rule files are introduced.

Assessment Metrics	52
Writing Recommended Rule Checks	52
Saving Error Measurements as Properties	53
Coding Complex Design Rule Checks	54
Creating a Property to Reflect the Yield Impact of an Individual Error.....	57
Writing Expressions for Scores and Metrics	58
Writing Rules That Display in DFM Report Card.....	59
Writing Rule Decks For Automatic Layout Enhancement	71
Doubling Vias	71
Expanding Enclosures.....	75
Expanding Enclosures After Doubling Vias.....	76
DFM Rules and Metrics Verification.....	77

Assessment Metrics

Analyzing a design using Calibre YieldAnalyzer is a two-step process.

1. Identify problem areas in your design.
2. Calculate [Quality Assessment Metrics](#) or [Yield Assessment Metrics](#) based on the number, size, and severity of the problems identified in Step 1.

Several tasks are common to assessing the manufacturability of a design.

Writing Recommended Rule Checks	52
Saving Error Measurements as Properties	53
Coding Complex Design Rule Checks	54
Creating a Property to Reflect the Yield Impact of an Individual Error	57
Writing Expressions for Scores and Metrics	58

Writing Recommended Rule Checks

Unlike design rule checks, which define hard and fast rules that should not be violated, recommended rule checks differ from Calibre nmDRC rules in that they define rules that can be violated, but at a cost.

Recommended rules typically come in groups, defining ranges of design configurations.

Procedure

1. Write the basic rule check using standard SVRF operations. Often, you can model your recommended rule off an existing Calibre nmDRC rule.

Existing Calibre nmDRC rule used as a template:

```
EXT gate <0.5 ABUT >0<90 OPPOSITE
```

Recommended Rule:

```
EXT gate range ABUT >0<90 OPPOSITE
```

2. Define the general ranges of values that are of interest to you.

Table 3-1. Sample Spacing Ranges

Mode	Range	Description
Calibre nmDRC	<0.5	Not allowed
RRA	>=0.5 <0.51	Terrible
RRA	>=0.51 <0.52	Very Bad
RRA	>=0.52 <0.53	Bad

Table 3-1. Sample Spacing Ranges (cont.)

Mode	Range	Description
RRA	$\geq 0.53 < 0.54$	Not too bad
RRA	$\geq 0.54 < 0.55$	Pretty good
Not Checked	≥ 0.55	Excellent

3. Create one recommended rule for each range:

```
Gate_Space_1 = EXT gate >=0.5 <0.51 ABUT >0<90 OPPOSITE
Gate_Space_2 = EXT gate >=0.51 <0.52 ABUT >0<90 OPPOSITE
Gate_Space_3 = EXT gate >=0.52 <0.53 ABUT >0<90 OPPOSITE
Gate_Space_4 = EXT gate >=0.53 <0.54 ABUT >0<90 OPPOSITE
Gate_Space_5 = EXT gate >=0.54 <0.55 ABUT >0<90 OPPOSITE
```

4. If you want your users to sort through analysis results using the DFM Report Card in Calibre RVE for DFM, then refer to “[DFM Report Card](#)” on page 409.

```
gate.space#rra#b.51 = EXT gate >=0.5 <0.51 ABUT >0<90 OPPOSITE
gate.space#rra#b.52 = EXT gate >=0.51 <0.52 ABUT >0<90 OPPOSITE
gate.space#rra#b.53 = EXT gate >=0.52 <0.53 ABUT >0<90 OPPOSITE
gate.space#rra#b.54 = EXT gate >=0.53 <0.54 ABUT >0<90 OPPOSITE
gate.space#rra#b.55 = EXT gate >=0.54 <0.55 ABUT >0<90 OPPOSITE
```

Saving Error Measurements as Properties

Use this method when you want to save the actual measurements from DFM or Calibre nmDRC checks, such as the exact width, space or common run length of a spacing error, as properties on individual geometries.

This method extends the usefulness of standard Calibre nmDRC checks, which by themselves tell you only that an error exists. It allows you to use Calibre nmDRC check measurement values in subsequent calculations.

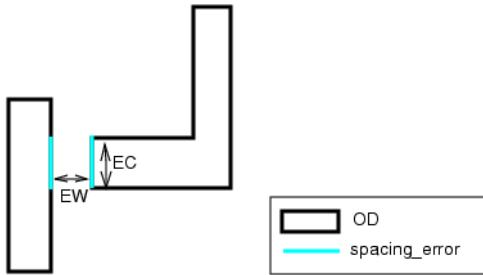
Procedure

1. Write the rule checks to find errors. For example:

```
spacing_error = External OD < 0.05
```

2. Identify important properties. For example:

```
Common_run_length = EC(spacing_error)
Spacing_meas = EW(spacing_error)
```



3. Use **DFM Property** to create an annotated error layer, tracking these properties. For example:

```
annotated_error = DFM PROPERTY spacing_error
[Common_run_length = EC(spacing_error)]
[Spacing_meas = EW(spacing_error)]
```

4. Use **DFM RDB** to write the annotated layer, complete with properties, to a results database. For example:

```
DFM RDB annotated_error errors.rdb SAME CELL
```

Coding Complex Design Rule Checks

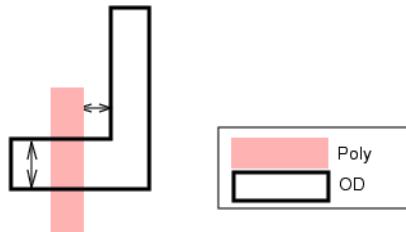
Complex design rule checks identify problems caused by the interaction of multiple conditions that might otherwise be benign.

These conditions might be inter-layer, intra-layer, or a combination of both.

Procedure

1. Define the conditions that are problematic. For example:

Assume the following geometries.



You might define a problem to exist when:

- Gate poly to L-shaped OD has spacing less than S. This is a sign that there may be a problem because you know this to cause the width of the gate to vary on the chip due to lithographic rounding of the inside corners of OD.
- Change in gate width varies by greater than 20%. This is a problem because it is a greater change than has been accounted for in simulations.

Assume you know that the width varies according to a formula such as:

$$\delta\text{Width} = \frac{K}{S}$$

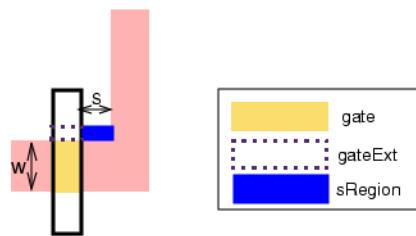
where K is a process dependent factor.

2. Create derived POLYGON layers containing the individual errors that contribute to the problem. For example:

The problem shown in step 1 might be written as:

```
gate = Poly AND OD
gate_l = gate COIN INSIDE EDGE OD
gate_ext = EXPAND EDGE gate_l OUTSIDE BY 0.0041
gate_s = EXT gate_ext OD < S REGION OPPOSITE
```

3. Design an equation to calculate the percentage by which the gate width changes. For example, assume the problem shown in step 1:



$$\text{PercentChange} = \frac{\delta\text{Width}}{\text{gateWidth}}$$

$$\delta\text{Width} = \frac{K}{S}$$

We can write this in terms of known quantities because we know that S is the AREA of the sRegion divided by its width, which is 0.004.

$$\deltaWidth = \frac{K}{(AREA(sRegion))/0.004} = \frac{K \times 0.004}{AREA(sRegion)}$$

We can calculate the gate width because we can use the AREA function to get the area of the gate and the length of the gate:

$$gateLength = (AREA(gateExt))/0.004$$

$$gateWidth = \frac{AREA(gate)}{(AREA(gateExt))/0.004} = \frac{AREA(gate) \times 0.004}{AREA(gateExt)}$$

So,

$$PercentChange = \frac{K \times 0.004}{AREA(sRegion)} \div \frac{AREA(gate) \times 0.004}{AREA(gateExt)}$$

or

$$PercentChange = \frac{K \times AREA(gateExt)}{AREA(gate) \times AREA(sRegion)}$$

4. Use [DFM Property](#) with constraints to isolate those portions of the design where all the conditions that are required to cause the problem exist. For example:

The property based on the *PercentChange* equation in step 3 might be written as:

```
problems = DFM PROPERTY gateExt sRegion gate OVERLAP ABUT ALSO
[- = (K*AREA(gateExt)) / (AREA(sRegion)*AREA(gate))] > 0.20
```

Note the use of the property name “-”, which instructs the operation to use the constraint as a filter for data written to the results layer, without saving the actual property value.

5. Use [DRC Check Map](#) to write the problem spots to a unique layer in the results database. For example:

```
problems {COPY problems}
DRC CHECK MAP problems 1146
```

Note

 Another option is to annotate the results layer with the property value, and use that property value for filtering results in Calibre RVE for DFM. The code for this option would be:

```
problems = DFM PROPERTY gateExt sRegion gate OVERLAP ABUT ALSO
[delta_w = (K*AREA(gateExt)) / (AREA(sRegion)*AREA(gate))]
> 0.20
DFM RDB problems problem_area.rdb SAME CELL
```

Creating a Property to Reflect the Yield Impact of an Individual Error

After you have identified errors within your design, you can generate a score for each error to see where potential improvements are likely to be most beneficial.

Procedure

1. Write the DFM rule checks to find errors. For example:

```
spacing_error = External OD < 0.05
```

2. Identify the properties that you can use to calculate the yield impact of a specific error. For example:

Assume the yield impact of an individual error can be represented as the following:

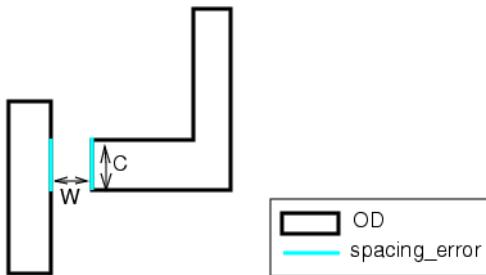
$$\text{impact} = \frac{KC}{W^2}$$

where:

K is a constant

C is Common_run_length = EC(spacing_error)

W is actual spacing = EW(spacing_error)



3. Use DFM Property to calculate the score for each error cluster. For example:

```
error_scores= DFM PROPERTY spacing_error
[dfm_index = (K*EC(spacing_error)/EW(spacing_error)^2]
```

4. Use DFM RDB to output the error clusters and their scores into an RDB. For example:

```
errors {DFM RDB error_scores "./output/scores.rdb" SAME CELL}
```

Note

 SAME CELL is required when you want to write the property values to the RDB along with the data.

5. Identify a formula that you can use to calculate the total yield impact of violations of this particular rule. For example:

Assume the total yield impact of violations is calculated as:

$$yield = \exp \left(-\sum \left(\frac{KC}{W^2} \right) \right)$$

6. Use [DFM Analyze](#) to calculate a total “yield impact” for the rule. In most cases, it is easiest for the designer if this data is written to the same RDB as the scores for the individual errors. For example:

```
total_impact { DFM ANALYZE spacing_error >= 0
  [EXP(-SUM( (K*EC(spacing_error))/(EW(spacing_error)^2) ))]
  RDB ONLY "./output/scores.rdb" }
```

After running the rule file on the target design, designers are able to do the following:

- a. Load the RDB into Calibre RVE for DFM.
- b. View the results of DFM Analyze to see the total yield impact for the rule.
- c. Sort by the property created using [DFM Property](#) (dfm_index in the example) to see how individual errors contribute to the total yield impact and plan how to fix the most serious errors.

Writing Expressions for Scores and Metrics

You generate scores and metrics by analyzing the results produced by recommend rules or critical area checks.

Scores and metrics must take the form of an expression to be analyzed by the [DFM Analyze](#) operation. If you have process and defect data available from the foundry, you can design [Yield Assessment Metrics](#) to predict actual yield. If you do not have this data available, you can design [Quality Assessment Metrics](#) that you can use to compare design styles and ensure that modifications truly improve manufacturability.

Procedure

1. Choose a yield or quality assessment model that best suits the rule and layer under investigation. For this example we use a weighted sum.
2. Assign a cost or weight to each of the ranges checked by recommended rules.

Table 3-2. Sample Weightings for Spacing Ranges

Layer	Range	Description
gate.space#rra#.51	$\geq 0.5 < 0.51$	0.000125
gate.space#rra#.52	$\geq 0.51 < 0.52$	8e-05

Table 3-2. Sample Weightings for Spacing Ranges (cont.)

Layer	Range	Description
gate.space#rra#.53	$\geq 0.52 < 0.53$	4.5e-05
gate.space#rra#.54	$\geq 0.53 < 0.54$	2e-05
gate.space#rra#.55	$\geq 0.54 < 0.55$	5e-06

3. Write a [DFM Analyze](#) check to calculate the score for the rule and layer under investigation:

```
Gate_Quality_Index{
    DFM ANALYZE gate.space#rra#.51 gate.space#rra#.52
        gate.space#rra#.53 gate.space#rra#.54
        gate.space#rra#.55 >=0
            [ ((COUNT(gate.space#rra#.51))*0.000125
                +(COUNT(gate.space#rra#.52))*8e-05
                +(COUNT(gate.space#rra#.53))*4.5e-05
                +(COUNT(gate.space#rra#.54))*2e-05
                +(COUNT(gate.space#rra#.55))*5e-06) ]
    RDB ONLY "quality_report.rdb"
}
```

4. For a complete discussion of expressions supported by DFM Analyze, refer to “[DFM Expressions](#)” in the *SVRF Manual*.

Writing Rules That Display in DFM Report Card

The Calibre RVE for DFM user interface is designed to display DFM check and score data in a report card format.

This format simplifies the task of interpreting DFM results and planning how to improve the design under investigation. The tabular format of the DFM Report Card provides designers with multiple views of results analysis as follows:

- **Table Rows** — Types of errors.
- **Table Columns** — Impact of the errors.
- **Tabbed Pages** — Locations of errors.

The DFM Report Card functionality is driven by a set of special-purpose (reserved) annotations—see “[DFM Report Card Controls](#)” on page 410 for complete information. In order for DFM results to display properly, the rule writer must insert commands that generate the required annotations. This topic describes how to add annotations to a DFM database, which annotations to use, and what values to use.

Prerequisites

- For best results you should make a working copy of the Calibre CFA tutorials and example kits (eKits).

Try It! 	Writing Critical Feature Analysis Rule Checks Tutorial and Example Kit Includes the documentation and data for writing CFA rule checks. The procedures provide a hands-on introduction to the concepts described in the “Assessment Metrics” on page 52 in the <i>Calibre YieldAnalyzer and YieldEnhancer User’s and Reference Manual</i> . Go to this page on Support Center to download the eKit (Documentation tab, Document Types=Getting Started Guide). The link goes to the latest release.
---	---

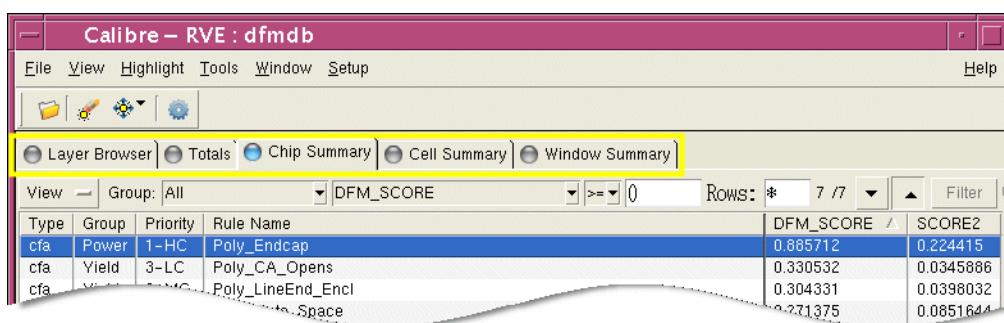
Try It! 	Calibre Critical Feature Analysis Tutorial and Example Kit Includes documentation and data for performing Critical Feature Analysis (CFA) with the Calibre YieldAnalyzer and Calibre YieldEnhancer tools. This tutorial walks you through building an analysis database, assessing design quality, locating and prioritizing fixes for the worst problems, and locating and prioritizing fixes for areas most likely to fail. Go to this page on Support Center to download the eKit (Documentation tab, Document Types=Getting Started Guide). The link goes to the latest release.
---	---

- At any time you can run the tutorials to see first hand how the annotations in the checks translate into report card elements.

Procedure

- Decide how you want to display scores to your user.

In the CFA tutorials and example kits, the annotations in the rules resulted in the following DFM Report Card:



- How many tabs do you need?

Each tab in the DFM Report Card provides a different way to investigate where errors occur. Of the tabs you see, only the **Layers** tab is visible by default. Each additional tab provide users a different way of reviewing check results:

- **Chip Summary** — Displays chip-wide scores for each check. It allows users to understand the overall impact of rule violations on manufacturability.
 - **Cell Summary** — Displays scores calculated on a cell-by-cell basis. It allows users to understand how individual cells impact manufacturability.
 - **Window Summary** — Displays scores calculated on a window-by-window basis. It allows users to identify the areas on the chip that are most likely to fail.
 - **Total** — Displays scores that summarize the impact of multiple related checks.
- b. How many columns do you need?

The columns displayed in the DFM Report Card provide designers with multiple ways to interpret the effect of the rule violations on yield and manufacturability. There are two types of columns: default and optional.

- **Default** — Sort mechanisms that help designers zero in on the data that is important to them. While these columns are always visible, they do not always contain data. It is up to the rule writer to supply the column data in the form of annotations.
- **Optional** — Metrics that report on the errors that were found. Metrics are simply expressions for calculating a score indicating the relative impact of the errors. Examples of some very simple metrics include:

Count — The number of individual errors found.

Area — The area affected.

Normalized Area — The percentage of the area under investigation that is affected.

Optional columns are represented on every tab except the **Layers** tab. The number of default columns shown in the report card varies according to the tab that is selected and the number of characteristics you choose to feature.

- **Total and Chip Summary** — Contain four default columns: Type, Group, Priority, and Rule Name.
- **Cell Summary** — Contains two default columns: Cell Name and Instance Count.
- **Window Summary** — Contains one default column: Window Id.

In addition, each of these tabs contain the same number of optional columns, generated as needed to display composite scores calculated based on characteristics you decide are important. In the case of the CFA tutorials, three optional columns

represent the scores before and after Calibre YieldEnhancer tools improved the design, plus a score showing actual improvement.

- c. What tips or explanations do you want to display?

The bottom of the report card contains a text box that displays the message associated with the selected check or score. This message is typically a description of a score or a suggested design modification that corrects a problem.

2. Write the checks that identify DFM errors.

A rule file that generates a report card-compliant DFM database can be viewed as containing two types of rule checks:

- The rule checks that identify errors.
- The rule checks that translate errors into scores.

As this tutorial is focused on displaying DFM results, not writing DFM checks, assume you have written the following rule:

```
width_distance = INTERNAL M1 <= 0.2
width_distance_prop = DFM PROPERTY width_distance
[DISTANCE = EW(width_distance)]
```

The first rule generates a new layer containing “errors” on M1 where a geometry has a width less than or equal to 0.2 microns. The second rule generates a copy of that layer with a property attached to each error, reporting on the actual width.

3. Use the [DFM RDB](#) command to write these results to the DFM database:

```
DFM RDB width_distance_prop "mydfm.rdb" ALL CELLS
```

Although this tutorial describes the steps for generating a DFM database, you must use the DFM RDB command to write geometries to that database. The DFM RDB command does not generate an RDB when the rule file is evaluated using calibre -dfm; it instead writes to the DFM database.

4. For each tab that you want to display in the DFM Report Card, write a rule check to calculate the scores. Assume you are interested in the number of errors found. The basic rule check takes the following form:

```
DFM ANALYZE width_distance [COUNT(width_distance) >= 0]
RDB ONLY DV COORD NULL
```

- a. If you want to display the **Chip Summary** tab, use [DFM Analyze](#) to calculate the full-chip score based on the check results:

```
DFM ANALYZE width_distance [COUNT(width_distance) >= 0]
// DFM Score for Chip
RDB ONLY DV COORD NULL
```

- b. If you want to display the **Cell Summary** tab, use DFM Analyze to calculate cell-by-cell scores based on the check results:

```
DFM ANALYZE [COUNT(width_distance) >= 0] BY CELL NOPSEUDO
// DFM Score by Cell
RDB ONLY DV COORD NULL
```

Notice that the commands are the same except that this one uses the BY CELL partitioning method.

- c. If you want to display the **Window Summary** tab, use [DFM Analyze](#) to calculate window-by-window scores based on the check results:

```
DFM ANALYZE [COUNT(width_distance) >= 0] WINDOW WIN_X WIN_Y
// DFM Score by Window
RDB ONLY DV COORD NULL
```

where WIN_X and WIN_Y are variables that define the window size in the x and y directions, respectively.

Note

 Notice that the commands are the same except that this one uses the WINDOW partitioning method.

The DFM RDB and DFM Analyze commands created in steps 3 and 4 form the basic code block you can work with for the remainder of this tutorial.

```
DFM RDB width_distance_prop "mydfm.rdb" ALL CELLS
DFM ANALYZE width_distance [COUNT(width_distance) >= 0]
// DFM Score for Chip
RDB ONLY DV COORD NULL
DFM ANALYZE width_distance [COUNT(width_distance) >= 0]
BY CELL NOPSEUDO
// DFM Score by Cell
RDB ONLY DV COORD NULL
DFM ANALYZE width_distance [COUNT(width_distance) >= 0]
WINDOW WIN_X WIN_Y
// DFM Score by Window
RDB ONLY DV COORD NULL
```

5. Copy the boilerplate statements below and paste them at the end of each of DFM RDB and DFM Analyze commands in your basic code block.

```
ANNOTATE [DFM_RULE = ""]
ANNOTATE [DFM_DESCRIPTION = ""]
ANNOTATE [DFM_LEVEL = ""]
ANNOTATE [DFM_TYPE = ""]
ANNOTATE [DFM_GROUP = ""]
ANNOTATE [DFM_PRIORITY = ""]
ANNOTATE [DFM_BIN = ""]
ANNOTATE [DFM_METRIC = ""]
```

Your code should now look like this:

```
DFM RDB width_distance_prop "mydfm.rdb" ALL CELLS
    ANNOTATE [DFM_RULE = ""]
    ANNOTATE [DFM_DESCRIPTION = ""]
    ANNOTATE [DFM_LEVEL = ""]
    ANNOTATE [DFM_TYPE = ""]
    ANNOTATE [DFM_GROUP = ""]
    ANNOTATE [DFM_PRIORITY = ""]
    ANNOTATE [DFM_BIN = ""]
    ANNOTATE [DFM_METRIC = ""]
DFM ANALYZE width_distance [COUNT(width_distance) >= 0]
    // DFM Score for Chip
    RDB ONLY DV COORD NULL
    ANNOTATE [DFM_RULE = ""]
    ANNOTATE [DFM_DESCRIPTION = ""]
    ANNOTATE [DFM_LEVEL = ""]
    ANNOTATE [DFM_TYPE = ""]
    ANNOTATE [DFM_GROUP = ""]
    ANNOTATE [DFM_PRIORITY = ""]
    ANNOTATE [DFM_BIN = ""]
    ANNOTATE [DFM_METRIC = ""]
DFM ANALYZE width_distance [COUNT(width_distance) >= 0]
    BY CELL NOPSEUDO
    // DFM Score by Cell
    RDB ONLY DV COORD NULL
    ANNOTATE [DFM_RULE = ""]
    ANNOTATE [DFM_DESCRIPTION = ""]
    ANNOTATE [DFM_LEVEL = ""]
    ANNOTATE [DFM_TYPE = ""]
    ANNOTATE [DFM_GROUP = ""]
    ANNOTATE [DFM_PRIORITY = ""]
    ANNOTATE [DFM_BIN = ""]
    ANNOTATE [DFM_METRIC = ""]
DFM ANALYZE width_distance [COUNT(width_distance) >= 0]
    WINDOW WIN_X WIN_Y
    // DFM Score by Window
    RDB ONLY DV COORD NULL
    ANNOTATE [DFM_RULE = ""]
    ANNOTATE [DFM_DESCRIPTION = ""]
    ANNOTATE [DFM_LEVEL = ""]
    ANNOTATE [DFM_TYPE = ""]
    ANNOTATE [DFM_GROUP = ""]
    ANNOTATE [DFM_PRIORITY = ""]
    ANNOTATE [DFM_BIN = ""]
    ANNOTATE [DFM_METRIC = ""]
```

6. Edit the annotations as follows:

- a. **DFM_RULE** — Supply a descriptive name for the rule. This is the same for all four of the commands, as each is merely a different representation of a single set of data.
- b. **DFM_DESCRIPTION** — Supply the text you want displayed to the user. This should be either a description of the check or a tip for fixing this type of error. This is the same for all four of the commands, as each is merely a different representation of a single set of data. If needed, you can supply multiple descriptions.

- c. **DFM_LEVEL** — Set this to one of: cell, chip, window, error, as follows:
 - o For the DFM RDB command that writes the errors to the DFM database, set DFM_LEVEL = “error”.
 - o For the DFM ANALYZE command that generates chip-wide scores, set DFM_LEVEL = “chip”.
 - o For the DFM ANALYZE command that generates cell-by-cell scores, set DFM_LEVEL = “cell”.
 - o For the DFM ANALYZE command that generates window-by-window scores, set DFM_LEVEL = “window”.
- d. **DFM_TYPE** — Set to a string that communicates the type of problem represented by these errors. This is the same for all four of the commands, as each is merely a different representation of a single set of data. This annotation must exist on the check data. If it is set to a null string, the Type column for this check is empty. In both tutorials, the type was the same for all errors: “cfa”.
- e. **DFM_GROUP** — Supply a string that is the name of a group to which this check belongs. This is the same for all four of the commands, as each is merely a different representation of a single set of data. When checks are assigned to groups using this annotation, users can sort or display by group. This annotation must exist on the check data. If it is set to a null string, the Group column is empty for this check.
- f. **DFM_PRIORITY** — Supply a meaningful string that assigns a priority to this check. These priorities provide another sort mechanism for the user. They reflect the relative importance of fixing this type of error compared to fixing other types of errors. This is the same for all four of the commands, as each is merely a different representation of a single set of data. This annotation must exist on the check data. If it is set to a null string, the Priority column is empty for this check.
- For these annotations, supply a short descriptive string. Note that the names of these annotations reflect the way they are used when reporting critical area analysis results.
- g. **DFM_METRIC** — Supply a name for the metric (or score). Metric names appear as column headers in the DFM Report Card.

Your code should now look like this:

```
DFM RDB width_distance_prop "mydfm.rdb" ALL CELLS
    ANNOTATE [DFM_RULE = "M1_narrow"]
    ANNOTATE [DFM_DESCRIPTION = "M1 with width less than 0.2"]
    ANNOTATE [DFM_DESCRIPTION = "Widen if possible"]
    ANNOTATE [DFM_LEVEL = "error"]
    ANNOTATE [DFM_TYPE = "width check"]
    ANNOTATE [DFM_GROUP = "metal problems"]
    ANNOTATE [DFM_PRIORITY = "high"]
    ANNOTATE [DFM_METRIC = "Simple Count"]
DFM ANALYZE width_distance [COUNT(width_distance) >= 0]
    // DFM Score for Chip
    RDB ONLY DV COORD NULL
    ANNOTATE [DFM_RULE = "M1_narrow"]
    ANNOTATE [DFM_DESCRIPTION = "M1 with width less than 0.2"]
    ANNOTATE [DFM_DESCRIPTION = "Widen if possible"]
    ANNOTATE [DFM_LEVEL = "chip"]
    ANNOTATE [DFM_TYPE = "width check"]
    ANNOTATE [DFM_GROUP = "metal problems"]
    ANNOTATE [DFM_PRIORITY = "high"]
    ANNOTATE [DFM_METRIC = "Simple Count"]
DFM ANALYZE width_distance [COUNT(width_distance) >= 0]
    BY CELL NOPSEUDO
    // DFM Score by Cell
    RDB ONLY DV COORD NULL
    ANNOTATE [DFM_RULE = "M1_narrow"]
    ANNOTATE [DFM_DESCRIPTION = "M1 with width less than 0.2"]
    ANNOTATE [DFM_DESCRIPTION = "Widen if possible"]
    ANNOTATE [DFM_LEVEL = "cell"]
    ANNOTATE [DFM_TYPE = "width check"]
    ANNOTATE [DFM_GROUP = "metal problems"]
    ANNOTATE [DFM_PRIORITY = "high"]
    ANNOTATE [DFM_METRIC = "Simple Count"]
DFM ANALYZE width_distance [COUNT(width_distance) >= 0]
    WINDOW WIN_X WIN_Y
    // DFM Score by Window
    RDB ONLY DV COORD NULL
    ANNOTATE [DFM_RULE = "M1_narrow"]
    ANNOTATE [DFM_DESCRIPTION = "M1 with width less than 0.2"]
    ANNOTATE [DFM_DESCRIPTION = "Widen if possible"]
    ANNOTATE [DFM_LEVEL = "window"]
    ANNOTATE [DFM_TYPE = "width check"]
    ANNOTATE [DFM_GROUP = "metal problems"]
    ANNOTATE [DFM_PRIORITY = "high"]
    ANNOTATE [DFM_METRIC = "Simple Count"]
```

7. To create additional columns, add another set of DFM Analyze operations, complete with annotations.

The code block in the previous step created one “optional” column, which displays the number of errors found. For this step, you create another column, this one reporting on the length of metal inside the area under investigation and having a width less than 0.2 microns.

- a. Decide how you want to calculate the score or metric.

To calculate the length, the basic operation is as follows:

```
DFM ANALYZE width_distance [EC(width_distance) >= 0]
    RDB ONLY DV COORD NULL
```

- b. Write one version of the DFM Analyze statement for each tab.

The actual operations used would be as follows:

```
DFM ANALYZE width_distance [EC(width_distance) >= 0]
    // DFM Score for Chip
    RDB ONLY DV COORD NULL
DFM ANALYZE width_distance [EC(width_distance) >= 0]
    BY CELL NOPSEUDO
    // DFM Score by Cell
    RDB ONLY DV COORD NULL
DFM ANALYZE width_distance [EC(width_distance) >= 0]
    WINDOW WIN_X WIN_Y
    // DFM Score by Window
    RDB ONLY DV COORD NULL
```

- c. Add the required annotations. Since this set of operations is reporting on the original error data, the only annotation you must change is the DFM_METRIC.

The final full set of code, complete with annotations, would look like this:

```

DFM RDB width_distance_prop "mydfm.rdb" ALL CELLS
    ANNOTATE [DFM_RULE = "M1_narrow"]
    ANNOTATE [DFM_DESCRIPTION = "M1 with width less than 0.2"]
    ANNOTATE [DFM_DESCRIPTION = "Widen if possible"]
    ANNOTATE [DFM_LEVEL = "error"]
    ANNOTATE [DFM_TYPE = "width check"]
    ANNOTATE [DFM_GROUP = "metal problems"]
    ANNOTATE [DFM_PRIORITY = "high"]
    ANNOTATE [DFM_METRIC = "Simple Count"]
DFM ANALYZE width_distance [COUNT(width_distance) >= 0]
    // DFM Score for Chip
    RDB ONLY DV COORD NULL
    ANNOTATE [DFM_RULE = "M1_narrow"]
    ANNOTATE [DFM_DESCRIPTION = "M1 with width less than 0.2"]
    ANNOTATE [DFM_DESCRIPTION = "Widen if possible"]
    ANNOTATE [DFM_LEVEL = "chip"]
    ANNOTATE [DFM_TYPE = "width check"]
    ANNOTATE [DFM_GROUP = "metal problems"]
    ANNOTATE [DFM_PRIORITY = "high"]
    ANNOTATE [DFM_METRIC = "Simple Count"]
DFM ANALYZE width_distance [COUNT(width_distance) >= 0]
    BY CELL NOPSEUDO
    // DFM Score by Cell
    RDB ONLY DV COORD NULL
    ANNOTATE [DFM_RULE = "M1_narrow"]
    ANNOTATE [DFM_DESCRIPTION = "M1 with width less than 0.2"]
    ANNOTATE [DFM_DESCRIPTION = "Widen if possible"]
    ANNOTATE [DFM_LEVEL = "cell"]
    ANNOTATE [DFM_TYPE = "width check"]
    ANNOTATE [DFM_GROUP = "metal problems"]
    ANNOTATE [DFM_PRIORITY = "high"]
    ANNOTATE [DFM_METRIC = "Simple Count"]
DFM ANALYZE width_distance [COUNT(width_distance) >= 0]
    WINDOW WIN_X WIN_Y
    // DFM Score by Window
    RDB ONLY DV COORD NULL
    ANNOTATE [DFM_RULE = "M1_narrow"]
    ANNOTATE [DFM_DESCRIPTION = "M1 with width less than 0.2"]
    ANNOTATE [DFM_DESCRIPTION = "Widen if possible"]
    ANNOTATE [DFM_LEVEL = "window"]
    ANNOTATE [DFM_TYPE = "width check"]
    ANNOTATE [DFM_GROUP = "metal problems"]
    ANNOTATE [DFM_PRIORITY = "high"]
    ANNOTATE [DFM_METRIC = "Simple Count"]
DFM ANALYZE width_distance [EC(width_distance) >= 0]
    // DFM Score for Chip
    RDB ONLY DV COORD NULL
    ANNOTATE [DFM_RULE = "M1_narrow"]
    ANNOTATE [DFM_DESCRIPTION = "M1 with width less than 0.2"]
    ANNOTATE [DFM_DESCRIPTION = "Widen if possible"]
    ANNOTATE [DFM_LEVEL = "chip"]
    ANNOTATE [DFM_TYPE = "width check"]
    ANNOTATE [DFM_GROUP = "metal problems"]
    ANNOTATE [DFM_PRIORITY = "high"]
    ANNOTATE [DFM_METRIC = "Common Run Length"]
DFM ANALYZE width_distance [EC(width_distance) >= 0]

```

```

    BY CELL NOPSEUDO
    // DFM Score by Cell
    RDB ONLY DV COORD NULL
    ANNOTATE [DFM_RULE = "M1_narrow"]
    ANNOTATE [DFM_DESCRIPTION = "M1 with width less than 0.2"]
    ANNOTATE [DFM_DESCRIPTION = "Widen if possible"]
    ANNOTATE [DFM_LEVEL = "cell"]
    ANNOTATE [DFM_TYPE = "width check"]
    ANNOTATE [DFM_GROUP = "metal problems"]
    ANNOTATE [DFM_PRIORITY = "high"]
    ANNOTATE [DFM_METRIC = "Common Run Length"]
    DFM ANALYZE width_distance [EC(width_distance) >= 0]
        WINDOW WIN_X WIN_Y
        // DFM Score by Window
        RDB ONLY DV COORD NULL
    ANNOTATE [DFM_RULE = "M1_narrow"]
    ANNOTATE [DFM_DESCRIPTION = "M1 with width less than 0.2"]
    ANNOTATE [DFM_DESCRIPTION = "Widen if possible"]
    ANNOTATE [DFM_LEVEL = "window"]
    ANNOTATE [DFM_TYPE = "width check"]
    ANNOTATE [DFM_GROUP = "metal problems"]
    ANNOTATE [DFM_PRIORITY = "high"]
    ANNOTATE [DFM_METRIC = "Common Run Length"]

```

8. Repeat steps 2 through 7 for each additional DFM check you include in your rule file.
9. Add annotations to the [DFM Database](#) statement to define chip totals.

Note

 For this tutorial, totals are not meaningful, as there is only one group and one rule check for each metric. In a production rule file, there can be many metrics and many groups, making these totals quite useful.

The easiest way to generate total scores and the **Totals** tab is to let the Calibre RVE for DFM Report Card calculate them for you. To do this, you annotate the database with one or more annotations named **DFM_METRIC_TOTAL_METHOD**.

This annotation required three arguments in a specific order:

(metric, calculation_method, cell_ref_total)

- **metric** — Must match the DFM_METRIC annotation associated with one or more checks. This defines which values to include in this total.
- **calculation_method** — Defines how totals should be calculated based on the per check scores for each of the checks.
- **cell_ref_total** — Defines how the report card calculates a total score for all the placements of a single cell. This is a second score calculated in addition to the **cell_score**, which is calculated using the **calculation_method** to combine the scores for each of the By Cell checks.

For more information about the DFM_METRIC_TOTAL_METHOD, refer to “[Database-Level Annotations](#)” on page 413.

Writing Rule Decks For Automatic Layout Enhancement

Calibre YieldEnhancer provides operations that make automatic modifications to your design. You can learn how to use these operations to perform certain design tasks.

Doubling Vias	71
Expanding Enclosures	75
Expanding Enclosures After Doubling Vias	76

Doubling Vias

The DFM Transition operation improves transitions by doubling vias where ever possible, thereby building redundancy into the design.

The DFM Transition operation can also insert rectangular vias that cover the original vias. See the complete usage syntax with examples for [DFM Transition](#) in the *Standard Verification Rule Format (SVRF) Manual*.

Try It! 	Calibre YieldEnhancer Via Tutorial and Example Kit Run the Calibre YieldEnhancer via enhancement flow. This flow is used to increase design robustness and reduce yield loss by improving the via transitions as part of the Calibre design for manufacturability (DFM) methodology. In the procedures, you use via doubling to insert second vias into a design and then you review the updated layout. Go to this page on Support Center to download the eKit (Documentation tab, Document Types=Getting Started Guide). The link goes to the latest release.
---	--

Prerequisites

- Connectivity is defined for your layout. If not, refer to “[Defining Connectivity](#)” on page 43.

Procedure

1. Make sure your DFM rule file includes the following statements and operations required for accessing connectivity data in the annotated GDS file:

```
DRC CELL TEXT YES
LAYOUT PROPERTY TEXT 126
TEXT LAYER 40
```

Note

 Note that this step is needed only if you have an annotated GDS file and intend on back-annotating your changes.

2. Define incremental connectivity (order-dependent) between the layers:

```
DRC INCREMENTAL CONNECT YES
CONNECT M1 M2 BY VIA1
CONNECT M2 M3 BY VIA2
CONNECT M3 M4 BY VIA3
CONNECT M4 M5 BY VIA4
...
```

3. Define the [DFM Spec Via](#) statements to be used by DFM Transition USEVIAS.

Via shapes can be square or rectangular, and for some layers, both square and rectangular via shapes are defined. DFM Transition uses the DOUBLEVIAS mode for square vias and the COVERVIAS mode for rectangular vias. For example, the following statements specify both square and rectangular via shapes in user units:

```
// Square via
DFM SPEC VIA VIA1.EN.1sq
    VIASHAPE .065
    ENC1 .035 0
    ENC2 .035 0

// Square via
DFM SPEC VIA VIA1.EN.2sq
    VIASHAPE .065
    ENC1 .016
    ENC2 .016

//Rectangular via
DFM SPEC VIA VIA1.EN.1re
    VIASHAPE .130 .065
    ENC1 .035 0
    ENC2 .035 0
...
```

4. Specify the [DFM Transition](#) operations used to generate the transition data.

In order to do DRC filtering and shorts avoidance, you must specify three invocations of DFM Transition for the *same* via layer, one for the below metal layer, one for the above metal layer, and one for the via. The three invocations of DFM Transition are processed concurrently. The results of the output (specified by OUTPUT1, OUTPUT2, OUTPUT3) are three derived layers that can be used for the necessary DRC checking

between the processing of each via layer. Three invocations of DFM Transition are shown in the following statements:

```

trans_M1 = DFM TRANSITION M1 M2 VIA1
    // Specify VIA names
    USEVIAS VIA1.EN.1sq VIA1.EN.2sq
    // M1 and M2 spacing
    SPACE1 0.065 METALSPACE 0.1 0 0.090 ENDTOEND 0.09 (-0.02) \
        0.08 ENDTOLINE 0.09 (-0.02) 0.08
    SPACE2 0.075 METALSPACE 0.1 0 0.090 ENDTOEND 0.09 (-0.02) \
        0.08 ENDTOLINE 0.09 (-0.02) 0.08
    // VIA1 spacing
    SPACE3 0.065 EUCLIDIAN 0.075 DBLSPACE 0.075
    // M1 and M2 jog constraint
    MINEDGE1 0.0325
    MINEDGE2 0.0325
    // Via layer enclosure (VIA1-1 and VIA1+1)
    VIAENC1 VIA0 GOOD 0.035 0 0.035 0
    VIAENC2 VIA2 GOOD 0.035 0 0.035 0
    // M1 and M2 minimum enclosed area
    MINENCAREA1 0.16 MOVE
    MINENCAREA2 0.16 MOVE
    // Output only the paired via shape
    NODUPLICATE
    // Double square vias
    DOUBLEVIAS
    // Generate output from M1
    OUTPUT1

trans_M2 = DFM TRANSITION M1 M2 VIA1
    // Specify VIA names
    USEVIAS VIA1.EN.1sq VIA1.EN.2sq
    // M1 and M2 spacing
    SPACE1 0.065 METALSPACE 0.1 0 0.090 ENDTOEND 0.09 (-0.02) \
        0.08 ENDTOLINE 0.09 (-0.02) 0.08
    SPACE2 0.075 METALSPACE 0.1 0 0.090 ENDTOEND 0.09 (-0.02) \
        0.08 ENDTOLINE 0.09 (-0.02) 0.08
    // VIA1 spacing
    SPACE3 0.065 EUCLIDIAN 0.075 DBLSPACE 0.075
    // M1 and M2 jog constraint
    MINEDGE1 0.0325
    MINEDGE2 0.0325
    // Via layer enclosure (VIA1-1 and VIA1+1)
    VIAENC1 VIA0 GOOD 0.035 0 0.035 0
    VIAENC2 VIA2 GOOD 0.035 0 0.035 0
    // M1 and M2 minimum enclosed area
    MINENCAREA1 0.16 MOVE
    MINENCAREA2 0.16 MOVE
    // Output only the paired via shape
    NODUPLICATE
    // Double square vias
    DOUBLEVIAS
    // Generate output from M2
    OUTPUT2

```

```
trans_VIA1 = DFM TRANSITION M1 M2 VIA1
    // Specify VIA names
    USEVIAS VIA1.EN.1sq VIA1.EN.2sq
    // M1 and M2 spacing
    SPACE1 0.065 METALSPACE 0.1 0 0.090 ENDTOEND 0.09 (-0.02) \
        0.08 ENDTOLINE 0.09 (-0.02) 0.08
    SPACE2 0.075 METALSPACE 0.1 0 0.090 ENDTOEND 0.09 (-0.02) \
        0.08 ENDTOLINE 0.09 (-0.02) 0.08
    // VIA1 spacing
    SPACE3 0.065 EUCLIDIAN 0.075 DBLSPACE 0.075
    // M1 and M2 jog constraint
    MINEDGE1 0.0325
    MINEDGE2 0.0325
    // Via layer enclosure (VIA1-1 and VIA1+1)
    VIAENC1 VIA0 GOOD 0.035 0 0.035 0
    VIAENC2 VIA2 GOOD 0.035 0 0.035 0
    // M1 and M2 minimum enclosed area
    MINENCAREA1 0.16 MOVE
    MINENCAREA2 0.16 MOVE
    // Output only the paired via shape
    NODUPLICATE
    // Double square vias
    DOUBLEVIAS
    // Generate output from VIA1
    OUTPUT3
```

Note

 DRC filtering is done here in a production rule deck. DRC filtering must be done on each layer before the next layer is done. For DRC filtering, the original shapes and the new shapes are merged to temporary layers, then DRC is run. Any new shapes interacting with error shapes are discarded. The resulting clean shapes are merged into the design and the vias are inserted in layer stack order.

5. Shorts can be created when the same layer is used in multiple DFM Transition operations without DRC filtering and merging. In the following example, metal2 is used in two DFM Transition operations to create trans_M2.1 and trans_M2.2. Since each operation uses metal2 independently, shorts can appear in areas of overlap when the result layers are OR'd together.

```
trans_M2.1 = DFM TRANSITION metal1 metal2 via1
...
trans_M2.2 = DFM TRANSITION metal2 metal3 via2
...
trans_M2 = trans_M2.1 OR trans_M2.2
```

6. To avoid this situation, first OR trans_M2.1 with metal2, then use the resultant layer as an input to the second DFM TRANSITION operation:

```
trans_M2.1 = DFM TRANSITION metal1 metal2 via1
...
trans_M2.2 = trans_m2.1 OR metal2
trans_M2 = trans_M2.2 metal3 via2
```

Expanding Enclosures

The DFM Expand Enclosure operation improves transitions by increasing the size of via enclosures, thereby reducing sensitivity to alignment issues.

Prerequisites

- Connectivity is defined for your layout. If not, refer to “[Defining Connectivity](#)” on page 43.

Procedure

1. Make sure your DFM rule file includes the following statements and operations required for accessing connectivity data in the annotated GDS file:

```
DRC CELL TEXT YES
LAYOUT PROPERTY TEXT 126
TEXT LAYER 40
```

Note

 Note that this step is needed only if you have an annotated GDS file and intend to back-annotate your changes.

2. Define connectivity between layers:

```
DRC INCREMENTAL CONNECT YES
CONNECT M1 M2 BY VIA1
```

3. Perform operations needed to derive the connection layers to be expanded.
4. For each connection layer to be expanded, use the [DFM Expand Enclosure](#) operation to create a derived layer containing the expansion geometries.

```
M1_EXP = DFM EXPAND ENCLOSURE VIA1 M1
        SIDE 0.04 LINE 0.04 END 0.05
        LINEENDMAX 0.045
        SPACE 0.14 STEP 0.005
```

5. Perform additional processing as needed.
6. Add connectivity data to the enclosure layers:

```
CONNECT M1 M1_EXP
```

7. Include one [DFM RDB](#) operation for each expanded enclosure layer to save it to a DFM RDB. Use the NODAL keyword to write connectivity data to the resulting database.

```
dfm_rdb_1_1 {
    DFM RDB M1_EXP "expand_enc.db"
    NOPSEUDO NOEMPTY
    NODAL
    MAXIMUM ALL}
```

Expanding Enclosures After Doubling Vias

Use this method when coding an SVRF rule file to expand enclosures for single vias that remain after doubling vias.

You must save the output from [DFM Transition](#) as derived layers so that the layers can serve as input to [DFM Expand Enclosure](#).

Prerequisites

- Connectivity is defined for your layout. If not, refer to “[Defining Connectivity](#)” on page 43.
- DFM Transition data is generated (double via insertion) and DRC filtering is done for your layout. If not, refer to “[Doubling Vias](#)” on page 71.

Procedure

1. Make sure your DFM rule file includes the following statements and operations required for accessing connectivity data in the annotated GDS file.

```
DRC CELL TEXT YES
LAYOUT PROPERTY TEXT 126
TEXT LAYER 40
```

Note

 Note that this step is needed only if you have an annotated GDS file and intend to backannotate your changes.

2. Define connectivity between layers.

```
DRC INCREMENTAL CONNECT YES
CONNECT M1 M2 BY VIA1
```

3. Perform operations needed to derive the connection layers to be expanded.

After [DFM Transition](#) via doubling and DRC filtering, assume that the resulting layers are M1_merged, M2_merged, and VIA1_merged.

```
//Select remaining single vias to expand
M1_and_M2_merged = M1_merged AND M2_merged
EXP_VIA1 = VIA1_merged INTERACT M1_and_M2_merged >=1 <2
```

4. Add connectivity data to the layers you use as input to [DFM Expand Enclosure](#).

```
CONNECT M1 M1_merged
CONNECT M2 M2_merged
CONNECT VIA1 EXP_VIA1
```

5. For each connection layer to be expanded, use the [DFM Expand Enclosure](#) operation to create a derived layer containing the expansion geometries.

```
M1_EXP = DFM EXPAND ENCLOSURE EXP_VIA1 M1_merged
SIDE 0.04 LINE 0.04 END 0.05
LINEENDMAX 0.045
SPACE 0.14 STEP 0.005
```

6. Perform additional processing as needed.
7. Add connectivity data to the expanded enclosure layers.
- CONNECT M1 M1_EXP
8. Include one **DFM RDB** operation for each expanded enclosure layer to save it to a DFM RDB. Use the NODAL keyword to write connectivity data to the resulting database.

```
dfm_rdb_1_1 {
    DFM RDB M1_EXP "expand_enc.db"
    NOPSEUDO NOEMPTY
    NODAL
    MAXIMUM ALL}
```

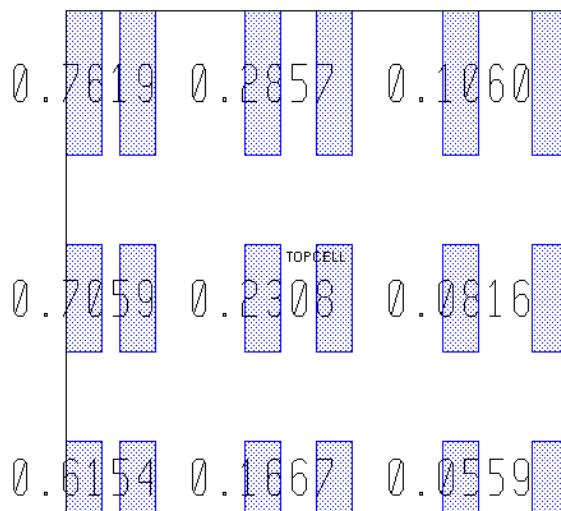
DFM Rules and Metrics Verification

Verifying a DFM rule deck ensures that the correct errors are being flagged (similar to a DRC run), and that the correct scores are being calculated for the violations.

In order to do this, you need to ensure that the values of properties produced in DFM Property operations are correct, which can be very time-consuming to perform by hand.

One method of expediting this process involves annotating a test layout with the expected property values, then reading in these values and comparing them using Calibre. Consider the following example:

Figure 3-1. Test Layout



On this layout, text has been added to each of the polygon structures that specify their expected scores. The associated text layer is mapped in the DFM rule deck:

```
//////////  
// ORIGINAL LAYER MAPPING  
//////////  
LAYER m1      1    // metal 1  
LAYER m1_txt  11   // metal 1 text  
//////////  
// TEXT LAYER MAPPING  
//////////  
LAYER MAP 1 TEXTTYPE 1 11
```

The DFM scores are then calculated:

```
m1_ext = EXT m1 <= 0.2 OPPOSITE  
m1_score = DFM PROPERTY m1_ext m1_txt_merged OVERLAP ABUT ALSO  
[dfm_score = ROUND(1-1/((0.001*EC(m1_ext)/EW(m1_ext)^3)+1),0.0001)]  
m1_score_check {  
    DFM RDB m1_score "cfa.rdb"  
}
```

Next, verification code is used to test the accuracy of the DFM score calculations. This code can be wrapped in an IF_DEF construct to run only when performing quality assurance testing.

```
//////////  
// QA routine to check the M1 Space DFM score  
//////////  
m1_txt_unmerged = DFM TEXT m1_txt PROPERTY NUMBER textprop  
m1_txt_merged = DFM PROPERTY MERGE m1 m1_txt_unmerged  
[qa_value = ROUND(PROPERTY(m1_txt_unmerged,textprop,1),0.0001)]  
m1_score_qa = DFM PROPERTY m1_score m1_txt_merged OVERLAP ABUT ALSO  
[dfm_score = PROPERTY(m1_score,dfm_score)]  
[qa_score = PROPERTY(m1_txt_merged,qa_value)]  
[delta = PROPERTY(m1_score,dfm_score) -  
PROPERTY(m1_txt_merged,qa_value)]  
m1_score_qa_check {  
    DFM RDB m1_score_qa "cfa_qa.rdb"  
}
```

The results can be output to the same RDB for easy comparison. If a delta value is non-zero, the DFM score is calculated incorrectly for a particular geometry.

Figure 3-2. Comparing Layout Text to DFM Scores

#	delta	dfm_score	qa_score
✗ 1	0	0.615400	0.615400
✗ 2	0	0.705900	0.705900
✗ 3	0	0.761900	0.761900
✗ 4	0	0.166700	0.166700
✗ 5	0	0.230800	0.230800
✗ 6	0	0.285700	0.285700
✗ 7	0	0.0559000	0.0559000
✗ 8	0	0.0816000	0.0816000
✗ 9	0	0.106000	0.106000

Chapter 4

Layout Enhancement

Calibre YieldEnhancer is a set of tools designed to improve yield.

You use Calibre YieldEnhancer to perform any of the following enhancements on your design:

- **Redundant Via Insertion** — Refers to replacing single vias with multiple vias where space allows. You can also specify to insert rectangular vias to cover existing square vias. For advanced technology nodes, these techniques can help improve the yield of a design by minimizing single via failures. See also [DFM Transition](#) and “[Doubling Vias](#)” on page 71.
- **Via Enclosure Expansion** — Refers to expanding the metal areas associated with vias. This technique can improve design reliability by reducing the sensitivity to opens caused by random particles. See also [DFM Expand Enclosure](#).
- **DFM Via Shift** — Refers to moving vias after RET biasing and retargeting to improve their enclosure by the intended upper and lower metals while observing mask rule check (MRC) spacing rules. See also [DFM Spec Via Shift](#), which provides the various MRC spacing rules required and is used by [DFM Via Shift](#).
- **Metal Fill** — A technique used to minimize the density variation across a layout, which reduces height variations during the CMP process. Adding metal fill to your design can reduce its sensitivity to design process variation and systematic defects. Calibre DFM SmartFill can generate fill with rectangles of various sizes and stretch factors, or with multiple polygon patterns, while adding a minimum number of fill shapes satisfying minimum, maximum and density gradient constraints. See also: [DFM Fill](#), [DFM Spec Fill](#), [DFM Spec Fill Shape](#), [DFM Spec Fill Optimizer](#).

Table 4-1 outlines the database formats that the Calibre YieldEnhancer flow supports.

Table 4-1. Supported Database Formats

Input (original) database	Output (enhanced) database
GDS	GDS
OASIS	OASIS
LEF/DEF	See “ fdIBA Command Line Syntax ” and “ fdIBA Examples ” in the <i>Calibre® Layout Comparison and Translation Guide</i>
OPENACCESS	

Stage 1: Defining Inputs for Via Insertion and Enclosure Expansion	86
Stage 2: Running YieldEnhancer to Add Vias and Expand Enclosures.	91
Stage 3: Backannotating Added Vias and Expanded Enclosures.	94
Stage 4: Defining Inputs for Adding Metal Fill	95
Stage 5: Running YieldEnhancer to Add Metal Fill.	96
Stage 6: Backannotating Added Metal Fill.	98
Sample Wrapper Script.	99

Calibre YieldEnhancer Flow

You can incorporate the YieldEnhancer flow into your design flow either by running a wrapper script or by running the six stages of the wrapper script independently.

Table 4-2 provides the stages of the flow.

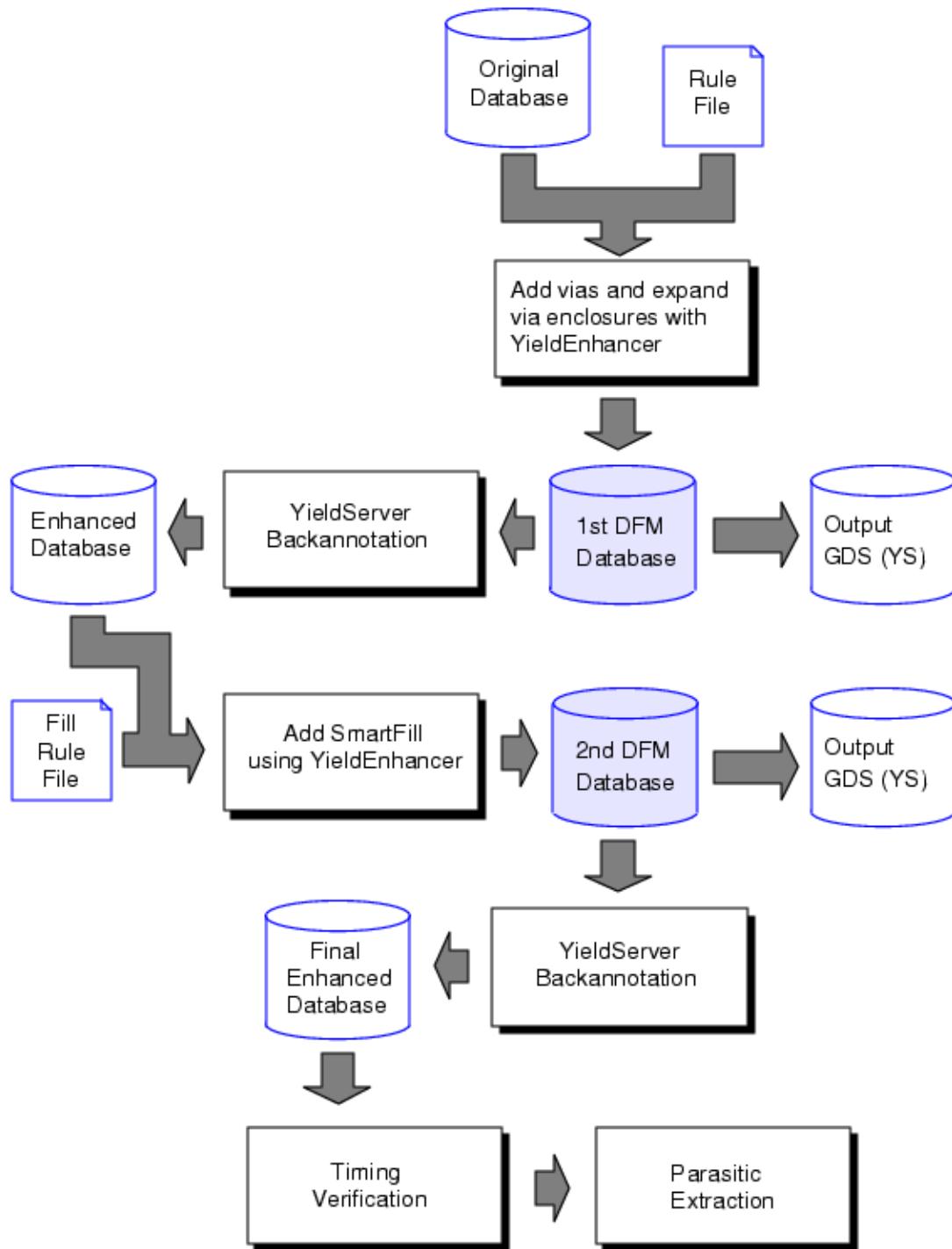
Table 4-2. Stages of Calibre YieldEnhancer Flow

Stage	Description	Inputs	Outputs
Stage 1: Defining Inputs for Via Insertion and Enclosure Expansion	Set up environment variables, set up required statements and paths in the rule deck, and create a layermap file.	Original rule deck.	Modified rule deck, layermap file.
Stage 2: Running YieldEnhancer to Add Vias and Expand Enclosures	Run Calibre YieldEnhancer in “calibre -dfm” mode to generate an output DFM database containing added vias and expanded enclosures.	Original design database, modified rule deck from stage 1, layermap file.	DFM database containing design modifications. If desired, a GDS file containing design modifications may be written using YieldServer.
Stage 3: Backannotating Added Vias and Expanded Enclosures	Run backannotation using the fdIBA utility to incorporate the added vias and expanded enclosures into the original design.	DFM database containing design modifications from stage 2, original design database, layermap file.	Enhanced design database (contains the original data plus design modifications from YieldEnhancer).
Stage 4: Defining Inputs for Adding Metal Fill	Set up environment variables, set up required statements and paths in the rule deck, and create a layermap file.	Original SmartFill rule deck.	Modified SmartFill rule deck, layermap file.
Stage 5: Running YieldEnhancer to Add Metal Fill	Run Calibre YieldEnhancer in “calibre -dfm” mode to generate an output DFM database containing metal fill.	Enhanced design database from stage 3, modified SmartFill rule deck, layermap file.	DFM database containing the original data plus design modifications from stages 2 and 5. If desired, a GDS file containing design modifications may be written using YieldServer.

Table 4-2. Stages of Calibre YieldEnhancer Flow (cont.)

Stage	Description	Inputs	Outputs
Stage 6: Backannotating Added Metal Fill	Run backannotation using the fdiBA utility to incorporate the metal fill into the original design.	DFM database containing the original data plus design modifications from stages 2 and 5, enhanced design database from stage 3, layermap file.	Final enhanced design database (contains the original data plus design modifications from YieldEnhancer).

Figure 4-1. Overview of Calibre YieldEnhancer Flow



Stage 1: Defining Inputs for Via Insertion and Enclosure Expansion

The first stage involves setting up your environment, preparing a rule file, and creating a layermap file prior to using Calibre YieldEnhancer to insert redundant vias and expand via enclosures.

Environment Variables for Input Database Files	86
Rule File Modifications	86
Layermap File	89

Environment Variables for Input Database Files

If you are using an input database in the LEF/DEF or OPENACCESS formats, you must set a few environment variables to prepare YieldEnhancer to read your database.

If you are using an input database in the GDS or OASIS formats, it is not necessary to set these environment variables.

- **MGC_CALIBRE_DB_READ_OPTIONS** — Defines various input options for reading the database.
- **LEF_PATHx** — Points to the location of a LEF file (if applicable), where *x* is an integer.
- **DEF_PATHx** — Points to the location of a DEF file (if applicable), where *x* is an integer.
- **TECH_LEF** — Points to the location of a LEF technology file, if applicable.

Figure 4-2. Setting Environment Variables for Input Database Files

```
# set env variables for the input database files
export MGC_CALIBRE_DB_READ_OPTIONS="-layerMap layer.map.256
    -outputBlockages -annotateNets PROPERTY 10"
export LEF_PATH1=./lef/tcbn65lp_7lm.lef
export DEF_PATH1=./def/test.def
```

Rule File Modifications

You must make modifications to your SVRF rule file to specify via insertion and expand via enclosure operations.

Procedure

1. Open your rule file in a text editor. If it is encrypted, note that there are two types of commands: encrypted and non-encrypted. The following steps apply to the non-encrypted section.

2. Specify the database type in a [Layout System](#) statement to match your input database, and define the path with [Layout Path](#). For example,

```
LAYOUT SYSTEM LEFDEF
LAYOUT PATH $LEF_PATH1 $DEF_PATH1
```

3. Modify the values of the [Layout Primary](#), [DRC Results Database](#), and [DRC Summary Report](#) statements as necessary to match your data and design environment.
4. Add SVRF statements to specify DRC checks, DFM checks, and the DFM database. For example:

```
GROUP DFM_CHECKS ?
GROUP DRC_CHECKS ?
DFM SELECT CHECK DFM_CHECKS
DRC SELECT CHECK DRC_CHECKS
DFM DATABASE dfmdb OVERWRITE
```

5. If you have your own via insertion deck, you can use the following steps to convert it for use in this flow. The goal here is to add extra code without changing the original via insertion deck in order to make both your via insertion and backannotation flow work. There are four suggested sections of script to add into the existing via insertion deck:
 - a. Add a #DEFINE keyword such as #DEFINE DB_BACK_ANNOTATION to enable Calibre to execute the YieldEnhancer backannotation-related script.
 - b. Add SVRF statements to enable net name case sensitivity.

```
#IFDEF DB_BACK_ANNOTATION
LAYOUT CASE YES
LAYOUT PRESERVE CASE YES
#ENDIF
```

- c. Add SVRF code to define layer mapping and distinguish between drawing layers and obstruction layers.

```
#IFDEF DB_BACK_ANNOTATION
LAYER MAP 31 DATATYPE 0 5001 // Mapping (31;0) to 5001 for M1i
LAYER MAP 31 DATATYPE 256 501 // Mapping (31;256) to 501 for M1i,
                                // fdi2gds convert LEF Obstruction
LAYER M1i_EXCL_OBS 5001 // Metal1 layer Exclude LEF Obstruction
..
..
LAYER MAP 51 DATATYPE 0 6001 // Mapping (51;0) to 6001 for VIA1i
LAYER MAP 51 DATATYPE 256 601 // Mapping (51;256) to 601 for
VIA1i,
                                // fdi2gds convert LEF Obstruction
LAYER VIA1i_EXCL_OBS 6001 // Define connect for M2 to M1, Exclude
                                // LEF Obstruction
#ENDIF
```

- d. Add connectivity and collect added via layers and nodal information with DFM RDB:

```
#IFDEF DB_BACK_ANNOTATION // COLLECT DB FOR DATA BACKANNOTATION
VIRTUAL CONNECT NAME "?"
DRC CELL TEXT YES
LAYOUT PROPERTY TEXT 10
TEXT LAYER 31 32 33 34 35 36 37 38 39
TEXT DEPTH PRIMARY
TEXT PRINT MAXIMUM 100
DRC INCREMENTAL CONNECT YES
//DFM TRANSITION
OAM1_V1 = (AM1 AND W1RightAM1V1a) AND AM1ForV1
OAM2_V1 = (AM2 AND W1RightAM2V1a) AND AM2ForV1
OAM2_V2 = (AM2 AND W1RightAM2V2a) AND AM2ForV2
..
OAM8_V7 = (AM8 AND W1RightAM8V7a) AND AM8ForV7
OAM8_V8 = (AM8 AND W1RightAM8V8a) AND AM8ForV8
OAM9_V8 = (AM9 AND W1RightAM9V8a) AND AM9ForV8
ATTACH 31 M1i_EXCL_OBS
..
ATTACH 39 M9i_EXCL_OBS
OAV1 = OAM1_V1 AND AVIA1
..
OAV8 = OAM8_V8 AND AVIA8
CONNECT M1i_EXCL_OBS M2i_EXCL_OBS BY VIA1i_EXCL_OBS
..
CONNECT M8i_EXCL_OBS M9i_EXCL_OBS BY VIA8i_EXCL_OBS
CONNECT M1i_EXCL_OBS OAM1_V1
CONNECT M2i_EXCL_OBS OAM2_V1
CONNECT M2i_EXCL_OBS OAM2_V2
..
CONNECT M8i_EXCL_OBS OAM8_V7
CONNECT M8i_EXCL_OBS OAM8_V8
CONNECT M9i_EXCL_OBS OAM9_V8
CONNECT OAM1_V1 OAM2_V1 BY OAV1
..
CONNECT OAM8_V8 OAM9_V8 BY OAV8
dfm_rdb_AVIA1 {
DFM RDB OAM1_V1 "topbo.db" OAM2_V1 OAV1 NOPSEUDO NOEMPTY
TRANSITION SAME CELL NODAL MAXIMUM ALL}
..
dfm_rdb_AVIA8 {
DFM RDB OAM8_V8 "topbo.db" OAM9_V8 OAV8 NOPSEUDO NOEMPTY
TRANSITION SAME CELL NODAL MAXIMUM ALL}
#endif
```

In this example script:

- Property 10 that stores net name information is treated as input layout database text using the Layout Property Text statement.
- OAM2_V1 is derived from AM2 (the AM2 layer is added metal2 from both VIA1 and VIA2 insertion; the AM2 layer is also a DRC-clean layer that is a derived layer from DFM Transition, or your own SVRF derivation, plus an

extended DRC check). OAM2_V1 is added M2 for VIA1 insertion. OAV1 is added VIA1; OAV1 is also DRC clean.

- Text is passed into M1i_EXCL_OBS (does not include LEF OBSTRUCTION), and connectivity is established by a Connect statement for M1i_EXCL_OBS, M2i_EXCL_OBS, VIA1i_EXCL_OBS ..., OAM1_V1, OAM2_V1, OAV1 and so on.
- Added via layers are collected as a tuple (added metal1 for via1 insertion “OAM1_V1”, added metal2 for via1 insertion “OAM2_V1”, and added via1 “OAV1”) containing nodal information with the DFM RDB command. The order of layers placed in the DFM RDB command must be as follows:

```
DFM RDB OAM1_V1 "topbo.db" OAM2_V1 OAV1 NOPSEUDO NOEMPTY
TRANSITION SAME CELL NODAL MAXIMUM ALL
```

6. If your input database format is LEF/DEF, change the values of Precision and Resolution to appropriately correspond to the DATABASE MICRONS value in the LEF file and the UNITS DISTANCE MICRONS value in the DEF file.

For example, if both values in the LEF and DEF files are 1000, the database unit in the converted GDS should be 1000. This translates into a Precision and Resolution ratio equal to 1000/5, or Precision 1000 and Resolution 5.

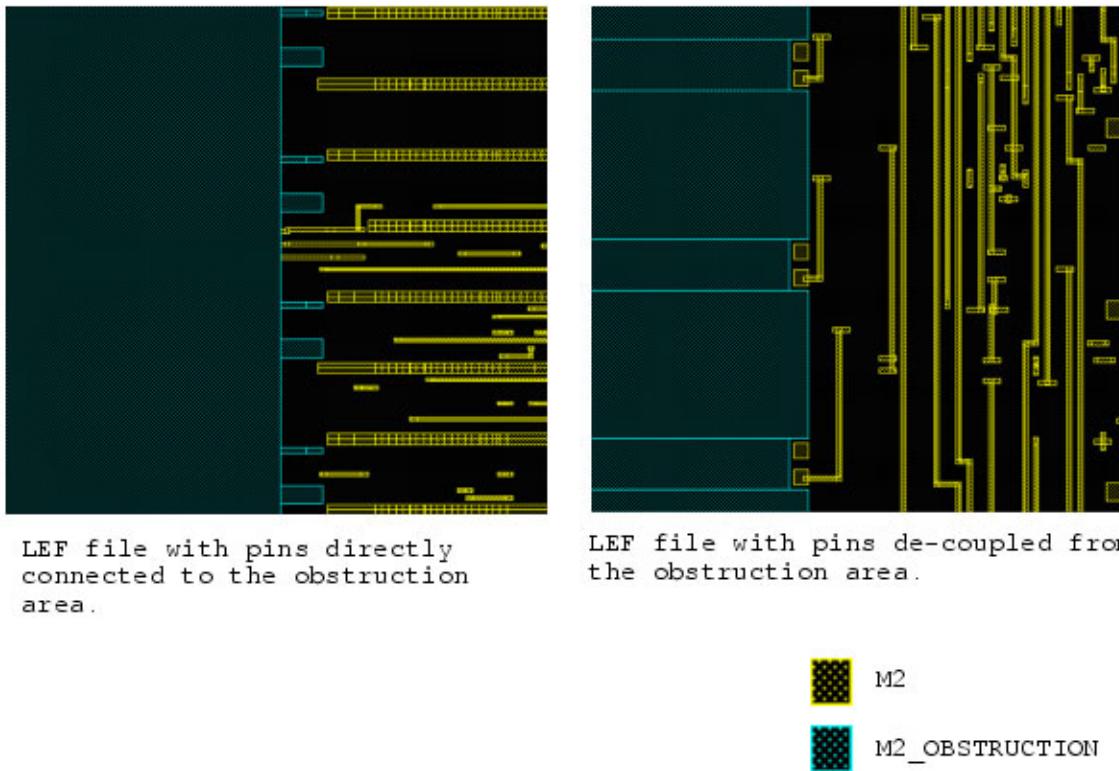
Alternatively, if one of the values in the LEF or DEF file is 2000, the database unit in the converted GDS should be 2000. This translates into a Precision and Resolution ratio equal to 2000/10, or Precision 2000 and Resolution 10.

7. Save and close the rule file.

Layermap File

If your input database format is LEF/DEF, you must provide a layermap file.

A layermap file is a mapping file that maps the layer names in the LEF/DEF format to layer numbers and data types recognized by the rule file used to perform layout enhancement. The layermap file must use the datatype feature to allow Calibre to distinguish between drawing layers and obstruction layers, and recognize the difference between pins that are directly connected to obstruction areas and pins that are de-coupled from obstruction areas (refer to [Figure 4-3](#)).

Figure 4-3. Connected Versus De-Coupled Pins

The following general-purpose layermap file shows how datatype distinguishing is done:

#layer name	layer purpose	layer#	data type
#DIFFi	drawing	1	0
#PO1		41	0
#CONT		39	0
M1	drawing	31	0
M1		31	256
VIA1	drawing	51	0
VIA1		51	256
M2	drawing	32	0
M2		32	256
VIA2	drawing	52	0
VIA2		52	256
M3	drawing	3	0
M3		33	256
VIA3	drawing	53	0
VIA3		53	256
M4	drawing	34	0
M4		34	256
VIA4	drawing	54	0
VIA4		54	256
M5	drawing	35	0
M5		35	256
VIA5	drawing	55	0
VIA5		55	256
M6	drawing	36	0
M6		36	256
VIA6	drawing	56	0
VIA6		56	256
M7	drawing	37	0
M7		37	256
VIA7	drawing	57	0
VIA7		57	256
M8	drawing	38	0
M8		38	256
VIA8	drawing	58	0
VIA8		58	256
M9	drawing	39	0
M9		39	256

In this file, the M2 (exact layer name from LEF/DEF) drawing layer is translated as layer 32 and data type 0, and the M2 obstruction region in LEF/DEF is translated as layer 32 and data type 256. Similarly, the VIA2 drawing layer is translated as layer 52 and data type 0, and the VIA2 obstruction region is translated as layer 52 and data type 256.

Stage 2: Running YieldEnhancer to Add Vias and Expand Enclosures

Stage 2 involves performing layout enhancement by running Calibre in -dfm mode with a Calibre YieldEnhancer rule file.

Procedure

1. Invoke Calibre from the command prompt as follows:

```
calibre -dfm -hier -turbo encrypted_rule_file.dfm >  
red_via_exp_enc.log
```

To obtain the fastest performance for large designs, use the hyper scaling option (-hyper).

2. You can optionally write out a GDS file from the generated DFM database by using Calibre YieldServer. The following script shows an example of how this can be done:

```

set layer_info_list [list \
    [list OAM1_V1 31 0] \
    [list OAM2_V1 32 0] \
    [list OAM2_V2 32 0] \
    [list OAM3_V2 33 0] \
    [list OAM3_V3 33 0] \
    [list OAM4_V3 34 0] \
    [list OAM4_V4 34 0] \
    [list OAM5_V4 35 0] \
    [list OAM5_V5 35 0] \
    [list OAM6_V5 36 0] \
    [list OAM6_V6 36 0] \
    [list OAM7_V6 37 0] \
    [list OAM7_V7 37 0] \
    [list OAM8_V7 38 0] \
    [list OAM8_V8 38 0] \
    [list OAM9_V8 39 40] \
    [list OAV1 51 0] \
    [list OAV2 52 0] \
    [list OAV3 53 0] \
    [list OAV4 54 0] \
    [list OAV5 55 0] \
    [list OAV6 56 0] \
    [list OAV7 57 0] \
    [list OAV8 58 40] \
]
## Initialize write_gds command:
set gds_cmd {}
lappend gds_cmd "dfm::write_gds"
lappend gds_cmd "-file"
lappend gds_cmd "ys_via_insertion.gds"
foreach linfo $layer_info_list {
    set layer [lindex $linfo 0]
    set cval [catch {dfm::get_layers $layer} emsg]
    if {$cval} {
        puts "Cannot find layer $layer in the database. Skipping..."
        continue
    } else {
        puts "Found layer $layer in the database."
    }
    ## Found the layer, add it to command:
    lappend gds_cmd "-layer_info"
    lappend gds_cmd $linfo
}
puts $gds_cmd
set cval [catch $gds_cmd emsg]
if {$cval} {
    error $emsg
}

```

To run this script, invoke Calibre as follows:

```
calibre -ys -dfmdb dfmdb -exec gds_out.tcl
```

where “dfmdb” is the name of your DFM database and “gds_out.tcl” is the name of the Calibre YieldServer script.

See the [Calibre YieldServer Reference Manual](#) for more information on Calibre YieldServer commands and scripts.

Stage 3: Backannotating Added Vias and Expanded Enclosures

Stage 3 backannotates the added redundant vias and modifies via enclosures into the original database to create an enhanced database.

Procedure

1. If your input database format is LEF/DEF, define a layermap file for backannotation. For example:

```
#calibre_layer LEFDEF_layer layer_type
OAM1_V1 M1 vialayer
OAM2_V1 M2 vialayer
OAV1 VIA1 vialayer
OAM2_V2 M2 vialayer
OAM3_V2 M3 vialayer
OAV2 VIA2 vialayer
OAM3_V3 M3 vialayer
OAM4_V3 M4 vialayer
OAV3 VIA3 vialayer
OAM4_V4 M4 vialayer
OAM5_V4 M5 vialayer
OAV4 VIA4 vialayer
OAM5_V5 M5 vialayer
OAM6_V5 M6 vialayer
OAV5 VIA5 vialayer
OAM6_V6 M6 vialayer
OAM7_V6 M7 vialayer
OAV6 VIA6 vialayer
OAM7_V7 M7 vialayer
OAM8_V7 M8 vialayer
OAV7 VIA7 vialayer
OAM8_V8 M8 vialayer
OAM9_V8 M9 vialayer
OAV8 VIA8 vialayer
DM1Fx M1 filllayer
DM2Fx M2 filllayer
DM3Fx M3 filllayer
DM4Fx M4 filllayer
DM5Fx M5 filllayer
DM6Fx M6 filllayer
DM7Fx M7 filllayer
DM8Fx M8 filllayer
DM9Fx M9 filllayer
```

2. To ensure that Calibre YieldEnhancer does not negatively impact any critical nets, you can instruct the tool to backannotate only changes to non-critical nets. To do this, you must:
 - Provide a critical nets text file listing the names of the critical nets. This file contains the name of each net on a separate line, and does not contain header text.
 - Pass this file to the **fdiBA** utility using the option **-criticalnetsfile** followed by the pathname to the critical nets file.
3. To run backannotation for the design database, type fdiBA at the command prompt. The command displays the full syntax describing multiple input options. Once you know which options you want to use, type the full command at the command prompt. Here is an fdiBA example syntax:

```
fdiBA -system LEFDEF -design ./def/test.def
      -defout ./test_DFM_fdiBA.def -dfmdb dfmdb
      -layermap fdiBA_lefdef_layer.map -logfile fdiBA_banno.log
```

The backannotation produces a transcript file (*fdiBA_banno.log*) in the fdiBA example syntax as well as the new DEF database (*test_DFM_fdiBA.def*) in the fdiBA example syntax. The transcript file reports the number of vias added to each net. Any errors or warnings, or notifications of unsuccessful backannotation for a specific via shape is included in the transcript.

Stage 4: Defining Inputs for Adding Metal Fill

Stage 4 defines the inputs for adding metal fill.

This stage is identical to “[Stage 1: Defining Inputs for Via Insertion and Enclosure Expansion](#)” on page 86, with two exceptions:

Procedure

1. If you are following the stages in sequence, modify the environment variables to read the enhanced database from “[Stage 3: Backannotating Added Vias and Expanded Enclosures](#)” on page 94.

```
export MGC_CALIBRE_DB_READ_OPTIONS="-layerMap layer.map
      -outputBlockages"
export DEF_PATH2=./test_DFM_fdiBA.def
```

2. If your input database format is LEF/DEF, create a layermap file. Unlike the layermap file from Stage 1, this layermap file should not distinguish between drawing and

obstruction data because running DFM Fill does not require net or connectivity information. For example:

#layer	name	layer_purpose	layer#	data	type
#DIFFi		drawing	1	0	
#PO1			41	0	
#CONT			39	0	
M1		drawing	31	0	
M1			31	0	
VIA1		drawing	51	0	
VIA1			51	0	
M2		drawing	32	0	
M2			32	0	
VIA2		drawing	52	0	
VIA2			52	0	

Stage 5: Running YieldEnhancer to Add Metal Fill

Stage 5 adds metal fill by running Calibre in -dfm mode with a Calibre YieldEnhancer rule file.

Procedure

1. Invoke Calibre from the command prompt as follows:

```
calibre -dfm -hier -turbo mgc_smart_fill.dfm > metal_fill.log
```

2. You can optionally write out a GDS file from the generated DFM database by using YieldServer. The following script shows an example of how this can be done:

```

set layer_list [list DM1Fx DM2Fx DM3Fx DM4Fx DM5Fx DM6Fx DM7Fx DM8Fx
DM9Fx]
set layer_info_list [list]
set layer_number 0
set data_type 7
set autoref "-autoref"
foreach layer $layer_list {
    set cval [catch {dfm::get_layers $layer} emsg]
    if {$cval} {
        puts "Cannot find layer $layer in the database. Skipping."
        continue
    }
    if {$layer == "DM1Fx"} {
        set layer_number 31
    } elseif {$layer == "DM2Fx"} {
        set layer_number 32
    } elseif {$layer == "DM3Fx"} {
        set layer_number 33
    } elseif {$layer == "DM4Fx"} {
        set layer_number 34
    } elseif {$layer == "DM5Fx"} {
        set layer_number 35
    } elseif {$layer == "DM6Fx"} {
        set layer_number 36
    } elseif {$layer == "DM7Fx"} {
        set layer_number 37
    } elseif {$layer == "DM8Fx"} {
        set layer_number 38
    } elseif {$layer == "DM9Fx"} {
        set layer_number 39
    } else {}
    set linfo [list $layer $layer_number $data_type $autoref]
    lappend layer_info_list $linfo
}

# Now build the command
set gds_cmd {}
lappend gds_cmd "dfm::write_gds"
lappend gds_cmd "-file"
lappend gds_cmd "ys_metal_fill.gds"
foreach linfo $layer_info_list {
    lappend gds_cmd "-layer_info"
    lappend gds_cmd $linfo
}

puts $gds_cmd
set cval [catch $gds_cmd emsg]
if {$cval} {
    error $emsg
}

```

Stage 6: Backannotating Added Metal Fill

Stage 6 backannotates the added redundant vias and modifies via enclosures into the original database to create an enhanced database.

Procedure

1. If your input database format is LEF/DEF, define a layermap file for backannotation.
For example:

```
#calibre_layer LEFDEF_layer layer_type
DM1Fx M1 filllayer
DM2Fx M2 filllayer
DM3Fx M3 filllayer
DM4Fx M4 filllayer
DM5Fx M5 filllayer
DM6Fx M6 filllayer
DM7Fx M7 filllayer
DM8Fx M8 filllayer
DM9Fx M9 filllayer
```

DM1Fx represents the derived layer from DFM Fill that passed into DRC Check Map to be written to the results database. M1 represents the corresponding layer name in LEF/DEF. The “filllayer” is used to specify the backannotation layer type. This layermap file is the same as you have used for previous via enhancing backannotation; the layer mapping of layer type filllayer is used for this step.

You can also choose to generate a new DEF file that contains metal fill polygons only (without the layout). Use an input DEF file that contains very minimal information. For example:

```
VERSION 5.5 ;
NAMECASESENSITIVE ON ;
DIVIDERCHAR "/" ;
BUSBITCHARS "[]" ;
DESIGN SE ;
TECHNOLOGY typ_nom ;
UNITS DISTANCE MICRONS 1000 ;
DIEAREA ( 0 0 ) ( 650000 650000 ) ;
END DESIGN
```

2. To run backannotation for the design database, type fdiBA at the command prompt. The command displays the full syntax describing multiple input options. Once you know which options you want to use, type the full command at the command prompt. Here is an example of the fdiBA command syntax:

```
fdiBA -system LEFDEF -design ./test_DFM_fdiBA.def
    -defout ./test_DFM_FILL_fdiBA.def -dfmdb dfmdb.fill
    -layermap fdiBA_lefdef_layer.map -logfile fdiBA_fill_bano.log
```

The backannotation produces a transcript file (*fdiBA_fill_banno.log*) as well as the new DEF database (test_DFM_FILL_fdiBA.def) as shown in the fdiBA command syntax example.

If you ran the stages in sequence, the final output, test_DFM_FILL_fdiBA.def, contains redundant via, via enclosure, and metal fill data.

Sample Wrapper Script

If desired, you can run multiple stages of the YieldEnhancer flow by using a wrapper script.

The script shown in [Figure 4-4](#) is configured for a LEF/DEF layout and runs all six stages. To modify it for your use, change the following items:

- Path names for input files.
- Path names for output files.
- Values of Precision and Resolution.

Figure 4-4. Sample Wrapper Script

```
#!/bin/sh
#set env variables for using a foreign database as input
export MGC_CALIBRE_DB_READ_OPTIONS="-layerMap layer.map.256
    -outputBlockages -annotateNets PROPERTY 10"
export LEF_PATH1=./lef/tcbn65lp_7lm.lef
export DEF_PATH1=./def/test.def
#run YieldEnhancer by -dfm (redundant via & via enclosure expand)
calibre -dfm -hier -turbo
DFM_CalibreYE_65nm_9M_6X2Z.12b1.encrypt.dfm
    > mgc_lefdef_dfm_enhancer.log
#write GDS from dfmdb
calibre -ys -dfmdb dfmdb -exec ys_output_added_via_gds.tcl
#YieldServer back annotation
fdiBA -system LEFDEF -design ./def/test.def
-defout ./test_DFM_fdiBA.def -dfmdb dfmdb
-layermap fdiBA_lefdef_layer.map -logfile fdiBA_bano.log
#set env variables for using foreign database as input
export MGC_CALIBRE_DB_READ_OPTIONS="-layerMap layer.map
    -outputBlockages"
export DEF_PATH2=./test_DFM_fdiBA.def
#run metal fill and collect binary dfmdb.fill
calibre -dfm -hier -turbo mgc_smart_fill_by_density_constraint.svrf
    > metal_fill.log
#write GDS from dfmdb.fill
calibre -ys -dfmdb dfmdb.fill -exec ys_output_metalfill_gds.tcl
#YieldServer backannotation for metal fill
fdiBA -system LEFDEF -design ./test_DFM_fdiBA.def
-defout ./test_DFM_FILL_fdiBA.def -dfmdb dfmdb.fill
-layermap fdiBA_lefdef_layer.map -logFile fdiBA_fill_bano.log
```


Chapter 5

Critical Area Analysis to Predict Yield

Critical Area Analysis allows you to identify those areas on a chip that are most sensitive to failures caused by random particles.

Using the critical area analysis functionality in Calibre YieldAnalyzer, you can find these sensitive areas, calculate the probability that a failure will actually occur at each location, then use the data from each of these areas to predict the yield for the entire chip.

Try It!	Calibre Critical Area Analysis Tutorial and Example Kit
	<p>Includes documentation and data for performing Critical Area Analysis (CAA) with Calibre YieldAnalyzer. The procedures step you through running CAA on a chip, CAA with marker layers, CAA on a cell library, and CAA with No Defect Density (NDD).</p> <p>Go to this page on Support Center to download the eKit (Documentation tab, Document Types=Getting Started Guide). The link goes to the latest release.</p>

Calibre YieldAnalyzer and Critical Area Analysis	102
CAA Flow	103
Defect Density File Data	104
CAA Rule Deck Generation Through the CAA Tool.....	104
Review of Critical Area Analysis Results	104
CAA Analysis with Marker Layers	105
CAA Analysis with Exclusion Layers	106
CAA Analysis with Memory Redundancy.....	108
Running a CAA Memory Analysis.....	108
CAA Memory By-Window Analysis	109
Re-analyzing Memory Analysis Results	109
Performing Net-Based CAA	112
CAA Library Analysis.....	114
Placement Boundaries.....	114
Performing CAA What-If Analysis	115
CAA with No Defect Density.....	117
CAA NDD Flow	117
CAA NDD Metrics	118
Running a CAA NDD Analysis	119

Calibre YieldAnalyzer and Critical Area Analysis

Calibre YieldAnalyzer Critical Area Analysis (CAA) evaluates the interactions between actual layout data and a set of defect particle models to extract all possible locations where electrical *short* or *open* conditions could occur on the chip.

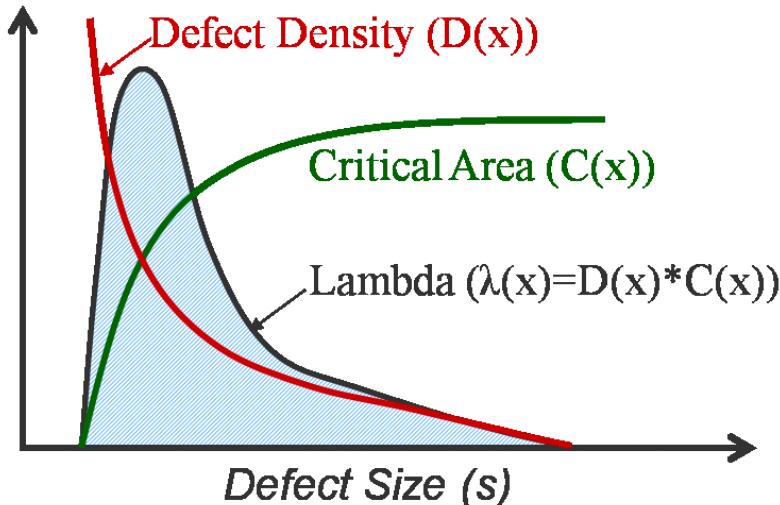
The set of defect particle models represent a range of possible particle sizes. Associated with each model is a *defect density* value, which represents the probability that particles of a particular size will be encountered during manufacturing. In most cases, defect particle models reflect actual defect size and density data provided by the chip foundry in an encrypted format.

Once the locations of potential shorts and opens have been identified, the tool calculates the expected silicon yield using the user-specified yield model. These are the two most commonly used yield models:

- Poisson
- Negative-Binomial with a clustering factor

These yield models are written in terms of an intermediate value, referred to as Lambda() or *weighted critical area*, which is calculated as the area of the shapes where the errors might occur multiplied by the defect density.

Figure 5-1. Defect Density, Critical Area, and Lambda



Lambda() represents the average number of fatal defects per unit area, or fault density. It is the goal of all concerned to reduce Lambda for any given design.

- Chip manufacturers attempt to reduce the number of introduced defects that negatively affect product yield. Expressed in terms of the image in [Figure 5-1](#), their goal is to move the Defect Density curve down and to the left.
- Chip designers attempt to reduce critical area. Expressed in terms of the image in [Figure 5-1](#), their goal is to move the Critical Area curve down and to the right.

Both these activities result in significant reduction in Lambda because the area under the Lambda curve decreases as the defect density curve moves down and to the left and the critical area curve down and to the right.

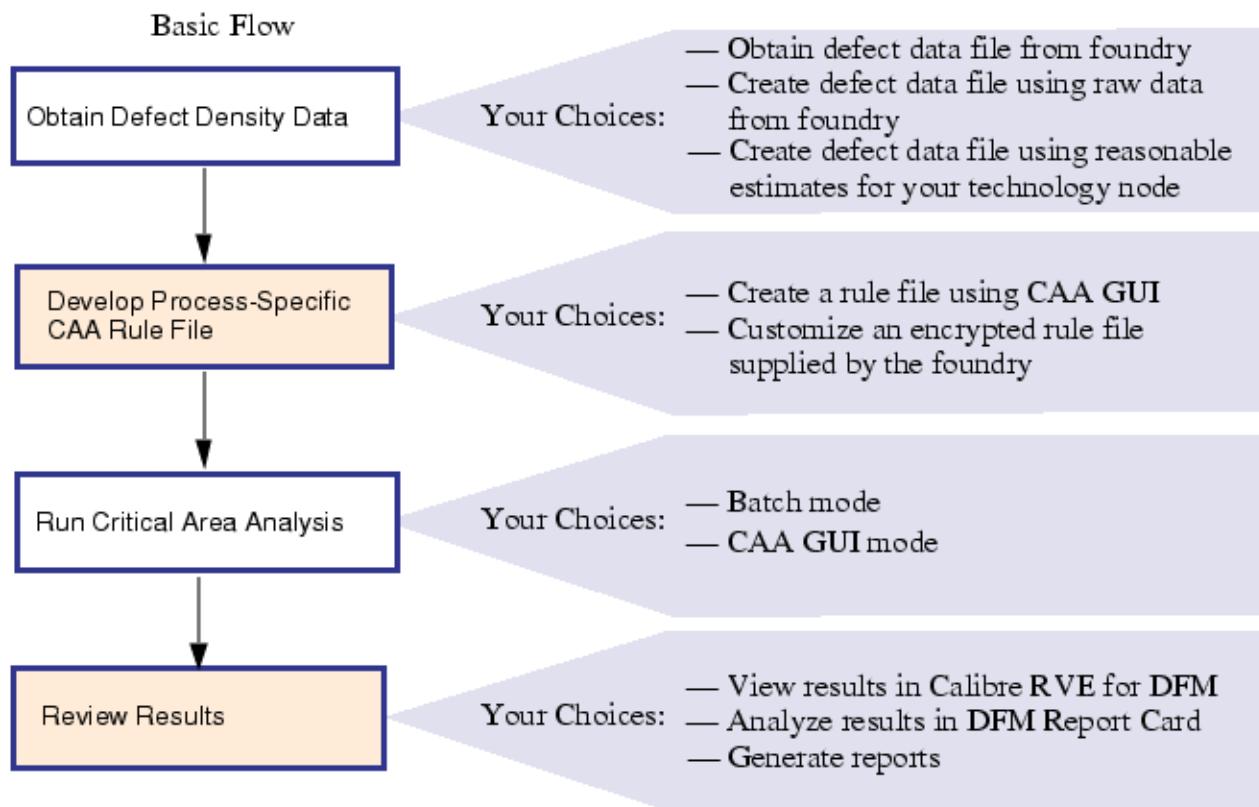
Calibre YieldAnalyzer CAA can also be used to predict via failure caused by random defects. Via failure analysis is based on via counts, which are extracted from actual layout data, and the via failure rate provided by the foundries. In general, foundries simplify via failure rates into single via failure per billion. For vias, **Lambda()** is the *weighted via failure*, and is calculated by multiplying single via counts and the single via failure rate. The same user-specified model used for calculating expected yield for opens and shorts can be used to obtain expected silicon yield related to random defects by single via failure.

CAA Flow

In general, the CAA flow with Calibre YieldAnalyzer consists of a basic flow with corresponding choices.

[Figure 5-2](#) outlines this basic flow.

Figure 5-2. Critical Area Analysis Flow Options Using Calibre



Defect Density File Data

Typically, defect density files are supplied by your foundry in encrypted form.

You may also supply an unencrypted defect density file. Defect density data must follow certain format specifications—see “[CAA Defect Data Format](#)” on page 350. If you do not have foundry defect density data available—see “[CAA with No Defect Density](#)” on page 117.

CAA Rule Deck Generation Through the CAA Tool

You can use Calibre Interactive - DFM to generate a CAA rule file and perform critical area analysis.

Review of Critical Area Analysis Results

Calibre provides several options for reviewing critical area analysis results.

These options include the following:

- **DFM Report Card** — Advanced analysis provided through the Calibre RVE for DFM user interface. This includes viewing data partitioned by cell or by window, and reported based on either hierarchical or flat analysis.
- **Yield Reports** — Generated by choosing the **View > Export to File** option in Calibre RVE for DFM.

CAA Analysis with Marker Layers

If your layout database contains marker layers embedded in different IP types, then these marker layers can be specified for special handling during CAA.

The marker layers are specified in the *caa_layer.txt* file, which is read by the Calibre Critical Area Analysis tool. The analysis is performed as before, but the reporting in Calibre RVE for DFM, and the batch reports (HTML, CVS, and text) now contain a DFM group for the marker layers. The chip report contains reports by layer and defect type for each marker layer.

This feature allows design and technology groups to collect yield loss data by IP type in addition to the usual hierarchical reporting in Calibre YieldAnalyzer. Typically, different marker layers are found in standard cells, RAMs, I/O pads, and analog or custom blocks.

Marker layers appear in the *caa_layer.txt* file have the following format:

```
// MARKER LAYER FOR CAA IS marker_layer
```

All marker layers are identified the same way, including derived layers. Note that the list of marker layers defined by the “// MARKER LAYER” lines is an ordered list. If a cell in the design is covered by more than one marker layer, then the first layer in the list has priority. The

shapes in the cell are only counted in the totals for the first marker layer covering the cell. The following is a section of an example *caa_layer.txt* file containing marker layers:

Figure 5-3. CAA Analysis with Marker Layers

```
...
LAYER BITCELLS 500 LAYER MAP 50 DATATYPE 0 500
// Original marker
// layers defined by
// LAYER and LAYER MAP
LAYER ANALOG 220 LAYER MAP 217 DATATYPE 3 220
LAYER STDCELLS 218 LAYER MAP 217 DATATYPE 1 218
...
BULK = EXTENT
HALO = BULK NOT ((BITCELLS OR ANALOG) OR STDCELLS)
// Derived marker
// layer
/////*BEGIN MAPPING CUSTOMER LAYERS TO MGC_CAA LAYERS////
// OD LAYER FOR CAA IS DIFF
// POLY1 LAYER FOR CAA IS PO
// CONT LAYER FOR CAA IS CO
...
// MARKER LAYER FOR CAA IS BITCELLS
// MARKER LAYER FOR CAA IS ANALOG
// MARKER LAYER FOR CAA IS STDCELLS
// MARKER LAYER FOR CAA IS HALO
/////*END OF MAPPING CUSTOMER LAYER TO MGC_CAA LAYERS///
```

CAA Analysis with Exclusion Layers

If your layout database contains exclusion layers embedded in different IP types, then specify the exclusion layers for special handling during CAA.

The exclusion layers are specified in the *caa_layer.txt* file, which is read by the Calibre Critical Area Analysis tool.

The format for this keyword is as follows:

// EXCLUDE LAYER FOR CAA IS *block_layer*

Similar to the MARKER keyword, the EXCLUDE keyword uses the exclusion layer's Calibre name.

For example, the following defines a layer as an exclusion layer:

```
// EXCLUDE LAYER FOR CAA IS blocklayer
```

You can also specify multiple exclusion layers. For example:

```
// EXCLUDE LAYER FOR CAA IS b1
// EXCLUDE LAYER FOR CAA IS b2
// EXCLUDE LAYER FOR CAA IS b3
```

See also “[Calibre Critical Area Analysis Tool](#)” on page 376.

CAA Analysis with Memory Redundancy

If your design contains embedded SRAMs with repair resources consisting of redundant rows or columns, then you can calculate the effect of memory redundancy on Defect-Limited Yield (DLY).

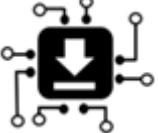
Running a CAA Memory Analysis.....	108
CAA Memory By-Window Analysis	109

Running a CAA Memory Analysis

This procedure demonstrates running CAA memory analysis with Calibre YieldAnalyzer CAA.

Prerequisites

- You have a valid memory configuration file—see “[Memory Configuration File Format](#)” on page 370.
- You have created a CAA runset.

Try It!	Calibre Critical Area Analysis Tutorial and Example Kit
	<p>Includes instructions on creating a runset along with documentation and data for performing Critical Area Analysis (CAA) with Calibre YieldAnalyzer. The procedures step you through running CAA on a chip, CAA with marker layers, CAA on a cell library, and CAA with No Defect Density (NDD).</p> <p>Go to this page on Support Center to download the eKit (Documentation tab, Document Types=Getting Started Guide). The link goes to the latest release.</p>

Procedure

1. From your working directory, start Calibre Interactive DFM:

```
calibre -gui -dfm
```

2. In the Load Runset File window, select the path to your runset and click **OK**.
3. Choose **Inputs > CAA > Memory Configuration File**, then enter the path or browse to your memory configuration file.

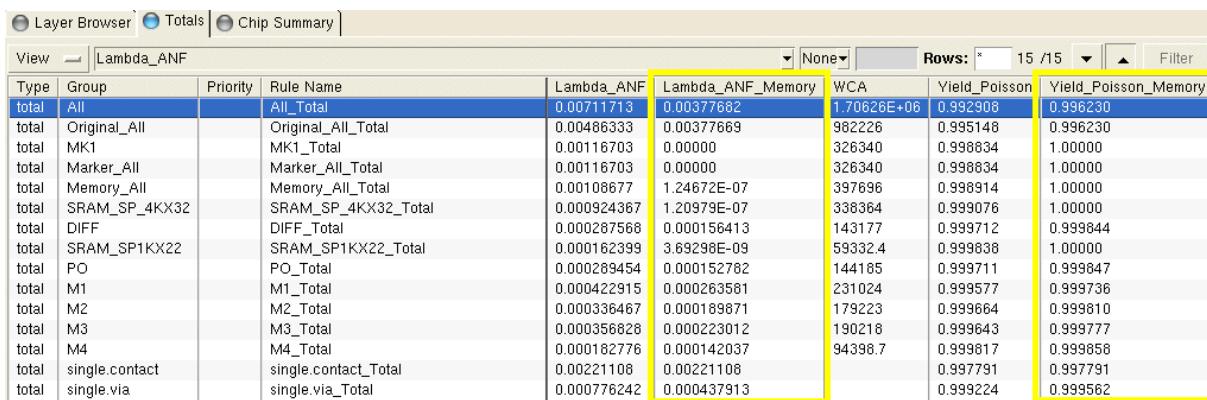
Note that the field for the memory configuration file path is not active for the [CAA NDD Flow](#), since memory analysis is not supported in that flow.

4. If you are running the GUI flow, click **Run CAA**.

If you are running Calibre Interactive DFM batch mode, choose **File > Save Runset As** to save the runset, then exit Calibre Interactive DFM. See “[Running Calibre Interactive in Batch Mode](#)” in the *Calibre Interactive User's Manual*.

- When the run completes, click **Start RVE** or open Calibre RVE for DFM from the command line if you did a Calibre Interactive DFM batch run.

Calibre RVE for DFM displays additional _Memory columns that show the results from the memory analysis.



Type	Group	Priority	Rule Name	Lambda_ANF	Lambda_ANF_Memory	WCA	Yield_Poisson	Yield_Poisson_Memory
total	All		All_Total	0.00711713	0.00377682	1.70626E+06	0.992908	0.996230
total	Original_All		Original_All_Total	0.00486333	0.00377669	982226	0.995148	0.996230
total	MK1		MK1_Total	0.00116703	0.00000	326340	0.998634	1.00000
total	Marker_All		Marker_All_Total	0.00116703	0.00000	326340	0.998634	1.00000
total	Memory_All		Memory_All_Total	0.00108677	1.24672E-07	397696	0.998914	1.00000
total	SRAM_SP_4KX32		SRAM_SP_4KX32_Total	0.000924367	1.20979E-07	338364	0.999076	1.00000
total	DIFF		DIFF_Total	0.000287568	0.000156413	143177	0.999712	0.999844
total	SRAM_SP1KX22		SRAM_SP1KX22_Total	0.000162399	3.69298E-09	59332.4	0.999838	1.00000
total	PO		PO_Total	0.000289454	0.000152782	144185	0.999711	0.999847
total	M1		M1_Total	0.000422915	0.000263581	231024	0.999577	0.999736
total	M2		M2_Total	0.000336467	0.000189871	179223	0.999664	0.999810
total	M3		M3_Total	0.000356828	0.000223012	190218	0.999643	0.999777
total	M4		M4_Total	0.000182776	0.000142037	94398.7	0.999817	0.999858
total	single.contact		single.contact_Total	0.00221108	0.00221108		0.997791	0.997791
total	single.via		single.via_Total	0.000776242	0.000437913		0.999224	0.999562

CAA Memory By-Window Analysis

CAA features in Calibre RVE for DFM enable you to generate by-window colormaps.

You can view and compare colormaps for Lambda_ANF_Memory and Yield_Memory metrics. Comparing the repaired versus unrepaired metrics allows you to see the improvements in the memory block in a clear, graphical way.

To view memory colormaps and histograms in a range corresponding to the non-memory metrics, choose **Setup > Options** and select the Histograms & Colormaps pane, then select the following options in the Data Distribution section:

- Choose Data Min:Max from the Range dropdown list
- Check the Normalize (CAA only) option

To leave the range values as is, without normalization, uncheck the Normalize (CAA only) option. The Normalize (CAA only) option forces the same range to be used for multiple colormaps and histograms.

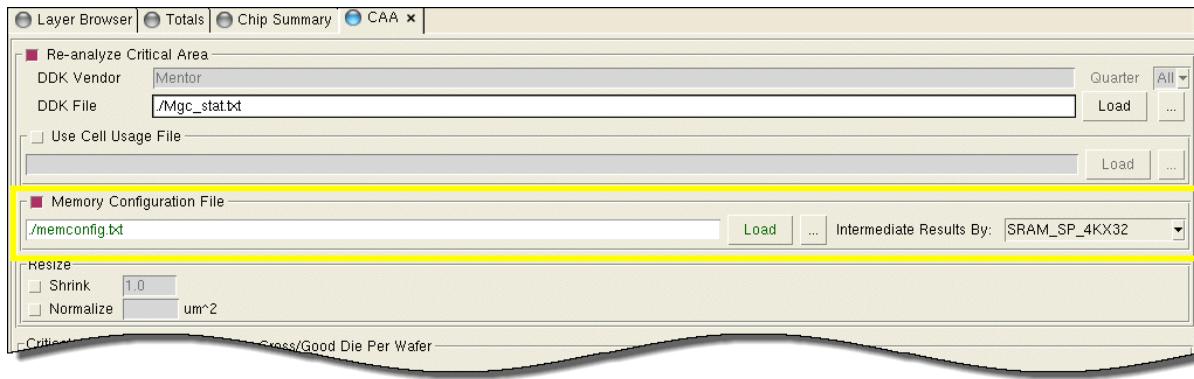
Re-analyzing Memory Analysis Results

Calibre RVE for DFM and Calibre DFM batch reporting support re-analysis for memory analysis results.

See “[DFM HTML Reporting](#)” in the *Calibre RVE User's Manual* for more information.

Procedure

- From Calibre RVE for DFM, choose **Tools > CAA**.
- Select the Re-analyze Critical Area option and check to then enable the Memory Configuration File option.



- Enter the path or browse to the memory configuration file.

Note that the field for the memory configuration file path is not active for the [CAA NDD Flow](#), since memory analysis is not supported in that flow.

- Click **Load**.
- Optionally, select a value from the Intermediate Results By dropdown list.

This enables the display of additional columns in Calibre RVE for DFM in the form *redundant_row:redundant_column*. The default value None disables the display of additional columns.

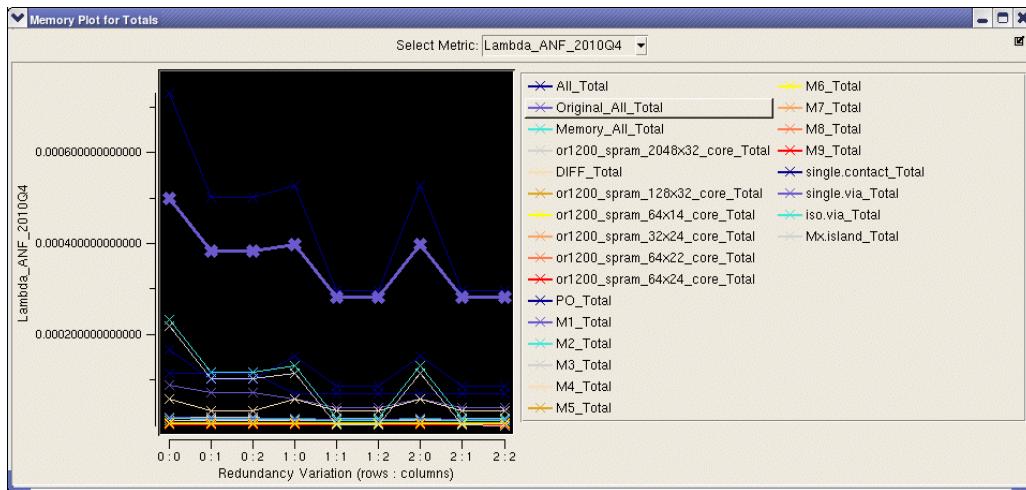
- Click **Analyze**. Additional intermediate results and repair ratio columns appear in Calibre RVE for DFM.

Type	Group	Priority	Rule Name	Lambda_ANF_2011Q3_Memory_0:0	Lambda_ANF_2011Q3_Memory_0:1	Lambda_ANF_2011Q3_Memory_0:2
total	All		All_Total	0.000257155	0.000249932	0.000249932
total	Origin...		Original_All_To...	0.000206134	0.000202523	0.000202523
total	Memor...		Memory_All_To...	5.10211E-05	4.74095E-05	4.74094E-05
total	or1200...		or1200_sram_...	4.11856E-05	4.11856E-05	4.11856E-05
total	DIFF		DIFF_Total	1.36096E-05	1.31358E-05	1.31358E-05
total	or1200...		or1200_sram_...	5.72861E-06	2.11699E-06	2.11698E-06
total	or1200...		or1200_sram_...	1.29796E-06	1.29796E-06	1.29796E-06
total	or1200...		or1200_sram_...	1.16168E-06	1.16168E-06	1.16168E-06
total	or1200...		or1200_sram_...	9.98995E-07	9.98995E-07	9.98995E-07
total	or1200...		or1200_sram_...	6.48230E-07	6.48230E-07	6.48230E-07
total	PO		PO_Total	5.20130E-05	5.20130E-05	5.20130E-05
				8.87341E-06		8.70346E-06

7. Select the **View > Plot Memory** option from the **Totals** or **Chip Summary** tabs to plot the yield model versus the redundancy variation. It is recommended that you first apply a filter to include only the rules you are interested in:

Type	Group	Priority	Rule Name	Value	Rows: * 6 /24 Filter
total	All			Lambda_ANF_2009Q4	Lambda_ANF_2009Q4
total	Original_All			0.000686298	0.000686298
total	Memory_All			0.000475763	0.000475763
total	or1200_sram_2048x32_core			0.000173326	0.000173326
total	PO			0.000107986	0.000107986
total	single.contact			0.000100954	0.000100954

8. After the filter is applied, choose **View > Plot Memory** to create the memory plot:



The following batch reporting key names can be used to re-analyze memory analysis results:

- AnalyzeCellUsageFile
- AnalyzeDDKFile
- AnalyzeMemoryConfigurationFile
- AnalyzeNormalizationArea
- AnalyzeShrinkFactor
- AnalyzeUmcPfxFile
- AnalyzeUmcPfxFilePassword
- AnalyzeYieldModels
- AnalyzeMemoryIntermediateResultsByCell.

Refer to “[Key Name Reference](#)” in the *Calibre RVE User's Manual* for more details on these key names.

Performing Net-Based CAA

You can perform a critical area analysis restricted to a specified list of critical nets.

Procedure

1. Modify the [CAA Layers File](#) to declare text layers. For example, if the net names of interest are on layer M1_TEXT, declare this layer as a text layer in the CAA layers file as follows:

```
LAYER M1 layer_number
LAYER M1_TEXT layer_number
TEXT LAYER M1_TEXT
ATTACH M1_TEXT M1
```

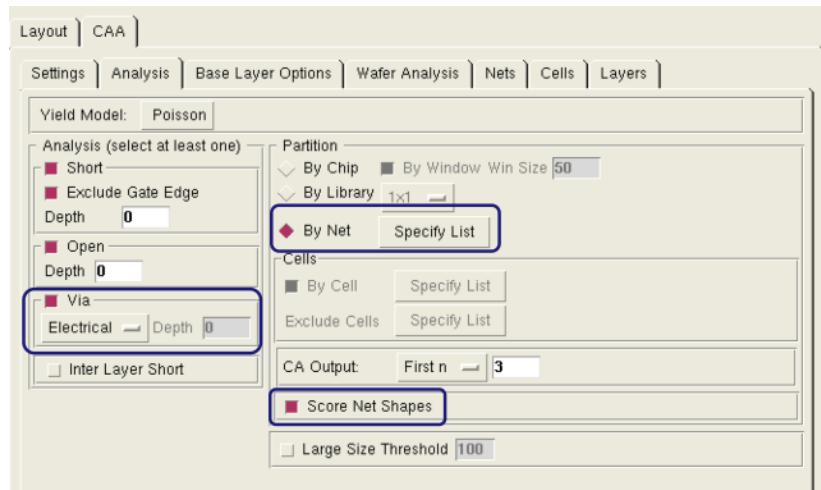
Alternatively, if text objects and geometries are both on the same layer, declare that layer as a text layer:

```
LAYER M1 layer_number
TEXT LAYER M1
```

2. Start Calibre Interactive DFM:

```
calibre -gui -dfm
```

3. In the **Inputs > CAA > Settings** tab, check the Use DFM CAF option.
4. In the **Analysis** tab, check Via and select Electrical from the dropdown list, and check By Net.

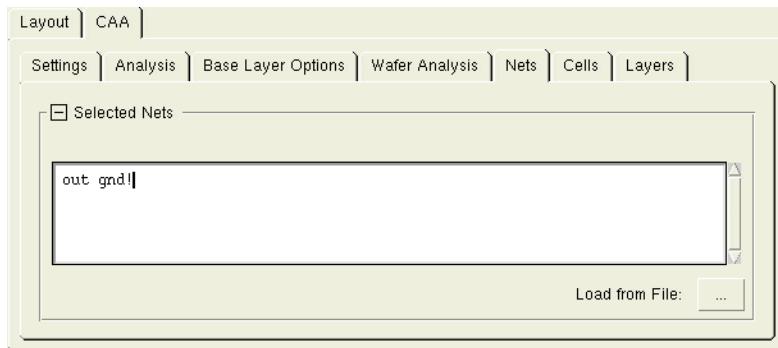


(Optional) You may also check Score Net Shapes. This switches on shape-based net score highlighting. This is only supported for the Mentor/GF/SEC DDK vendors.

To access the net scores per shape, do the following:

- a. When viewing results in Calibre RVE for DFM, click the **Drill Down** tab.

- b. Select a net rule.
 - c. Right-click the selected rule. Select one of two options that appear: Highlight WCA or Lambda_ANF.
 - d. The net shapes associated with the selected rule is highlighted with a new generated layer in your layout viewer.
5. Click **Specify List** and enter a list of critical nets (you can also load the list from a file by clicking **Load from File**).



6. (Optional) You can preserve the case of the net names by including the SVRF statement **LAYOUT PRESERVE NET CASE YES** in your runset as follows:
 - a. Click the DFM Options button on the left panel, or choose **Setup > DFM Options** if the button is not visible.
 - b. Click the **Include** tab and check **Include Rule Statements**.
 - c. Enter **LAYOUT PRESERVE NET CASE YES** in the text area.
 - d. Click **File > Save Runset** or **File > Save Runset As** to save the SVRF statement to your runset.
7. Click **Run CAA** on the left pane of the Calibre Interactive DFM GUI. When the analysis completes, a **Net Summary** tab is displayed in Calibre RVE for DFM that allows you to view results for and highlight each net.

Net Name	Lambda_ANF	WCA	Yield_Poisson
gnd	4.86052E-05	14275.1	0.999951
out	9.24454E-08	49.3597	1.00000

The NetSummary and NetDrillDown report types can be used to output similar results for DFM HTML Report. See “[DFM HTML Reporting](#)” in the *Calibre RVE User's Manual* for more information.

CAA Library Analysis

Within Calibre Interactive DFM, you can use the CAA library flow to quickly analyze a library of cells.

This flow generates a GDSII with a top cell that contains an array of cell instances for each analyzed cell in the library. Using this top cell, you view CAA results for the complete set of analyzed cells. For an array of instances, you view results that apply across cell boundaries. This is in addition to results within each cell. The results of the analysis are collected into a CSV file for use in spreadsheet programs. The CSV file facilitates sorting the results by various criteria to look for cells that need improvement.

Placement Boundaries 114

Placement Boundaries

To use the library flow, you must specify a layer that defines the cells' placement boundaries in a similar way as a boundary used for a place and route run.

You define the placement boundary as a layer in the *caa_layer.txt* file using the following syntax:

```
// FRAME LAYER FOR CAA IS layer_name
```

where *layer_name* is a layer name you have specified using the LAYER statement.

Alternatively, you can use *layer.datatype* numbers instead of the layer name. See the “[CAA Layers File](#)” on page 366.

Note

 Once you have defined the boundary layer with the FRAME LAYER element, you cannot use this in a LAYER MAP statement.

Use the following guidance when you define these boundaries:

- **Layout Database with Similar Cells** — Analysis is done on the entire library in one CAA run.
- **Standard Cells** — Analysis is done using a 3 x 3 grid of cells correctly abutted to account for edge interactions. It is crucial you use a correct, original frame layer that defines the placement box so that the cells share both contacts and power and ground rails. For Standard Cell library database files that have a top cell where one of each library cell is instantiated, the top cell is ignored.
- **IP Blocks** — Analysis is typically done 1 x 1 (stand-alone), or 1 x 3 (for I/O buffers). For larger IP blocks that do not need to be arrayed 3 x 3, this layer may be derived from an EXTENT or EXTENT CELL statement.

The library content may be predefined in a cell list file, otherwise the flow automatically generates a full list of all cells in the library GDS.

In the Calibre Interactive DFM CAA GUI, you must select By Library and By Cell and then specify the array selection, yield model, and so on. Selecting CA Output All is reasonable for standard cells, but can impact run time and database size on large IP blocks. The By Window selection is not useful on Standard Cells. To normalize the results to a unit area, enter a value in square microns for Die Area such as 1e6 to normalize to 1 mm².

Performing CAA What-If Analysis

The CAA What-if Analysis tool provides you with the ability see the effects of changes to a layer and defect type, or a new die size.

Specifically, you can do the following:

- Specify a percentage change for a layer and defect type and view the effect on the total Lambda, Yield, and Good Die Per Wafer.
- Specify a new die size and view the effect on Good Die Per Wafer.

Type	Group	Prio	Rule Name	Adjust_Lambda_%	Lambda_ANF	Single_Via_%	Total_Via_%	WCA	Yield_Pois	Yield_P
CAA	DIFF		OPEN	0.00000	6.48617E-06			0.9	Click	999
CAA	DIFF		SHORT	0.00000	8.47074E-06			0.9	Update	999
CAA	poly		OPEN	0.00000	1.00000E-05			0.9		998
CAA	poly		SHORT	0.00000	1.00000E-05			0.9		998
CAA	metal1		metal1.OPEN	0.00000	1.00000E-05			0.99971	0.999966	0.99998
CAA	metal1		metal1.SHORT	0.00000	1.00000E-05			14040.0	0.999969	0.99996
CAA	metal1		metal2.OPEN	0.00000	2.87681E-05			10480.4	0.999984	0.99998
CAA	metal1		metal2.SHORT	0.00000	2.87681E-05			14386.2	0.999971	0.99997
				0.00000	0.00000			6377.87	0.999992	0.99999

Procedure

- To access the What-if tool, from the **Chip Summary** tab in Calibre RVE for DFM, choose **View > What-if**.
- If you have an existing what-if configuration file, select Load from the What-if Configuration dropdown box—refer to “[CAA What-if Analysis Configuration File Format](#)” on page 373 for details on the format. Enter a path or browse to a valid configuration file and click **OK**.
- If you do not have an existing configuration file, enter new values (as positive or negative percentages) in the Adjust_Lambda_% column.
- Next to Die Width / Height, enter new values for the die width and height.

5. Click **Update**. What-if yield is calculated for each yield model and quarter, and displayed in added _What-if columns. Values are calculated only for the chip and total levels. (They are not calculated for the cell and window levels.)
6. If desired, save the configuration file by clicking **What-if Configuration > Save to File**.

CAA with No Defect Density

The Calibre YieldAnalyzer CAA No Defect Density (CAA NDD) flow acts as an automated DFM scoring tool that uses the Avg_Quality metric for scoring layers and rules. When you use the CAA NDD flow (supported for Calibre CAA tool versions 2017.2 and later), there is no need to provide foundry defect density data (D0). The standard analysis methods still apply (Chip, Cell, Window).

Try It! 	Calibre Critical Area Analysis Tutorial and Example Kit Includes documentation and data for performing Critical Area Analysis (CAA) with Calibre YieldAnalyzer. The procedures step you through running CAA on a chip, CAA with marker layers, CAA on a cell library, and CAA with No Defect Density (NDD). Go to this page on Support Center to download the eKit (Documentation tab, Document Types=Getting Started Guide). The link goes to the latest release.
---	---

CAA NDD Flow	117
CAA NDD Metrics.....	118
Running a CAA NDD Analysis.....	119
Reviewing CAA NDD Results.....	120

CAA NDD Flow

The CAA NDD flow allows you to run Calibre CAA as a DFM scoring tool using the standard CAA flow without the requirement of providing foundry defect density data or rule deck development.

The practice of predicting yield before tape-out can be complicated with the issue of exposing sensitive foundry defect density and yield data. Even when this data is made available, the extraction of accurate defect density (D0) data can be difficult and expensive for foundries. To address these complexities, the CAA NDD flow provides a simplified CAA flow that does not use foundry defect data. The CAA NDD flow uses CAA for a DFM scoring tool rather than a yield predictor. Additionally, the CAA NDD flow differs from the standard CAA flow in the following ways:

- The defect density data must be specified as “NA” in the CAA NDD defect file.
 - The CAA NDD flow replaces the standard defect density (D0) specification used for the CAA D0 Defect Data Format.
 - The CAA K-Factor Defect Data Format is not supported for the CAA NDD flow.

Note

 Refer to “[CAA Defect Data Format](#)” on page 350 for information on the CAA D0 Defect Data Format, CAA NDD Defect Data Format, CAA K-Factor Defect Data Format, and User-Defined List Format.

- The Yield Model selection in the Calibre Interactive DFM CAA GUI is disabled in the CAA NDD flow.
- An Avg_Quality metric is used for CAA NDD score results which are normalized for design comparison and IP checking.
- The Lambda and Yield values are suppressed in the CAA NDD output reports.
- The CAA Memory Analysis flow is not supported in the CAA NDD flow.

Along with these differences, the CAA NDD flow still has the following commonalities with the standard CAA flow:

- You need to determine the type of defects to simulate for your technology node.
- You need to specify the range of defect sizes:
 - Minimum defect size is determined by the design rule’s critical minimum dimension (CD).
 - Maximum defect size is usually an arbitrary value.

The CAA NDD flow, achieves the goal of DFM analysis to reduce the sensitivity of designs to manufacturing variability using average-based DFM scoring metrics without the issues of predicting yield.

CAA NDD Metrics

The CAA NDD flow uses an Avg_Quality metric that is a normalized value for a DFM score. This metric is suitable for design screening of chip and IP designs for DFM compliance that is consistent with quality and reliability initiatives.

The CAA NDD flow Avg_Quality metric is based on the ratio of $WCA/Area$ (the “kill ratio” for defects) and is defined for short and open checks as follows:

$$\text{Avg_Quality} = 1 - (WCA/Area)$$

where:

- WCA is the *weighted critical area*.
- $WCA/Area$ is always < 1 .
- $WCA/Area$ is a normalized value.

For via defects, the CAA NDD metric is defined as follows:

$$\text{Avg_Quality} = 1 - (\text{Single_Via\%}/100)$$

where:

- *Avg_Quality* for vias is based on the single via percentage in a design layout (*Single_Via%*).
- *Avg_Quality* for vias is equivalent to the via *Redundancy Ratio* where a higher value is a better score.

The *Avg_Quality* metric total score is an average, not a product, and is annotated to your rule file as follows:

```
ANNOTATE '[' DFM_METRIC_TOTAL_METHOD = "Avg_Quality,Average,Average" ']
```

Running a CAA NDD Analysis

You run a CAA NDD analysis from the Calibre Interactive DFM GUI or batch mode similar to a standard CAA run with the added advantage that defect density data is not required or used. The output from a CAA NDD run is a DFM score instead of a yield metric.

Note

 Certain CAA features are not available when running the CAA NDD flow (**Settings** tab: Memory Configuration File and the **Wafer Analysis** tab: Gross/Good Die Per Wafer).

Prerequisites

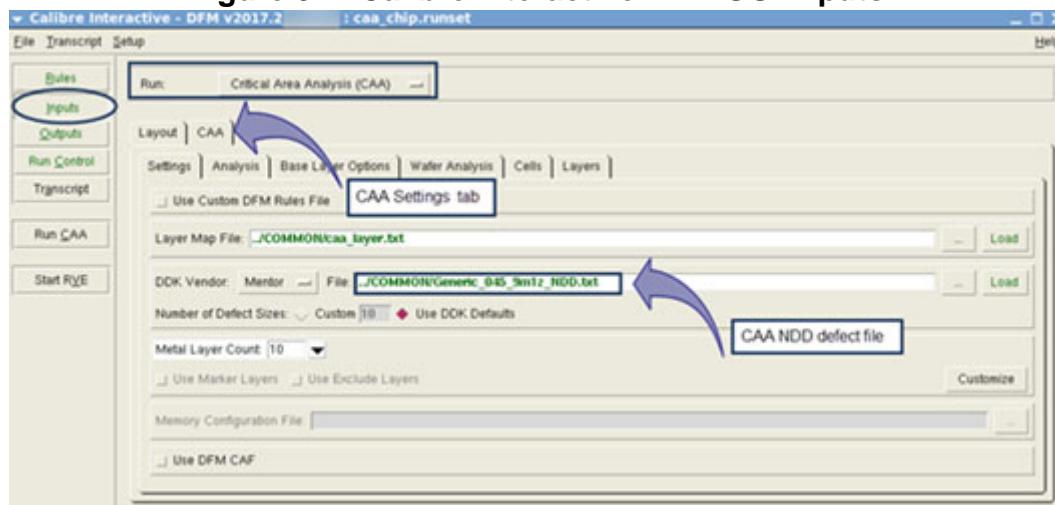
- You have access to a layout viewer that is integrated with Calibre Interactive, CAA from the Calibre Interactive DFM GUI, and Calibre RVE for DFM.
- In a text editor, inspect your CAA NDD defect file and verify that the values in the defect density column (D0) are specified as “NA”. For via analysis, the single via failure rate(s) should also be specified as “NA”.
- You have created a runset file for CAA that specifies the path to your CAA NDD defect file under the Calibre Interactive DFM GUI Inputs pane **CAA** tab.

Procedure

1. From the shell prompt, start your layout viewer and load the design layout.
2. Start Calibre Interactive DFM by choosing **Verification > Run DFM** in Calibre DESIGNrev, or use the **Calibre** menu in your supported layout viewer to select the appropriate command to run Calibre Interactive DFM.
3. In the Load Runset File dialog box, enter the full path to your runset in the Runset File Path and click **OK**.

4. Review the setup buttons on the left panel of the Calibre Interactive DFM GUI. The setup panes for each of the buttons contain the tabs and settings for your CAA run. Buttons for the setup panes are green if all information is available and red if information is missing.

Figure 5-4. Calibre Interactive DFM GUI Inputs



5. Click **Run CAA**. If the Overwrite File? dialog box appears, click **Don't Overwrite**. When your CAA NDD run completes, you can review the results in Calibre RVE for DFM.

Reviewing CAA NDD Results

You use the standard Calibre CAA options to review CAA NDD score metric results. The CAA NDD results are DFM scores instead of the yield results generated in a standard CAA run.

- DFM Report Card — Analyze and view flat and hierarchical data using Calibre RVE for DFM. In this user interface, data is partitioned by cell or by window, and can be displayed by colormap, histogram, plot, or highlighted in a connected layout viewer such as Calibre DESIGNrev or other supported layout viewer.
- Yield Reports — Export the data in Calibre RVE for DFM to a report format file for analysis. Note that Lambda or Yield metrics information is suppressed in the CAA NDD flow.

Prerequisites

- You have access to a layout viewer that is integrated with Calibre Interactive, CAA from the Calibre Interactive DFM GUI, and Calibre RVE for DFM.
- You have successfully completed the steps in “[Running a CAA NDD Analysis](#)” on page 119.

Procedure

1. Select the **Totals** tab in the Calibre RVE for DFM window and examine the CAA NDD total scores for each Group in the Avg_Quality results column as shown in [Figure 5-5](#).

Figure 5-5. CAA NDD Totals — Score Metric Results

Type	Group	Priority	Rule Name	Avg_Quality	WCA
total	All		All_Total	0.992743	110224
total	DIFF		DIFF_Total	0.950568	13808.6
total	poly		poly_Total	0.946961	14816.3
total	metal1		metal1_Total	0.892770	29954.3
total	metal2		metal2_Total	0.917549	23032.3
total	metal3		metal3_Total	0.931528	19127.3
total	metal4		metal4_Total	0.973828	7310.91
total	metal5		metal5_Total	0.993792	1734.21
total	metal6		metal6_Total	0.998456	431.440
total	metal7		metal7_Total	0.999994	1.71812
total	metal8		metal8_Total	0.999984	4.55009
total	metal9		metal9_Total	0.999995	1.48075
total	metal10		metal10_Total	0.999996	1.01378
total	single.contact		single.contact_Total	0.608933	
total	single.via		single.via_Total	0.949036	

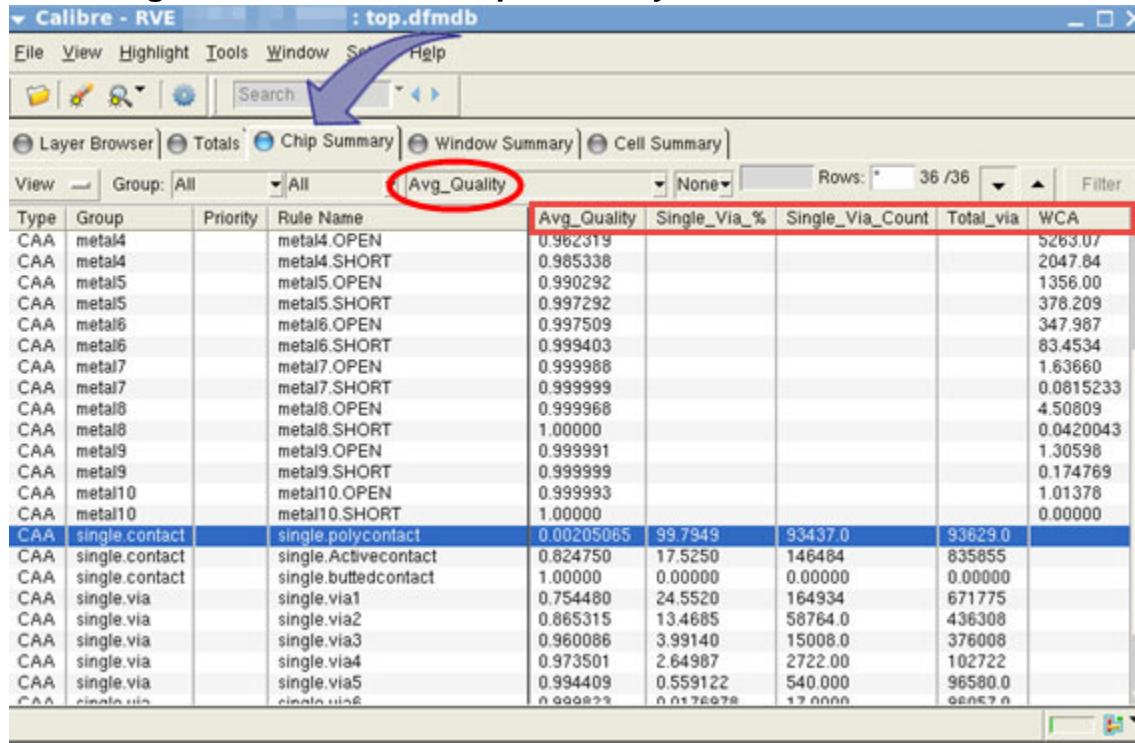
For comparison, examine the **Totals** tab yield metrics and results from a standard CAA run for each Group in the Calibre RVE for DFM window as shown in [Figure 5-6](#).

Figure 5-6. Standard CAA Totals — Yield Metric Results

Type	Group	Priority	Rule Name	Lambda_ANF	WCA	Yield_Poisson
total	All		All_Total	0.000401353	100822	0.999599
total	DIFF		DIFF_Total	2.12622E-05	10631.1	0.999979
total	poly		poly_Total	2.86121E-05	14306.1	0.999971
total	metal1		metal1_Total	5.22620E-05	27087.1	0.999948
total	metal2		metal2_Total	3.66800E-05	20872.4	0.999963
total	metal3		metal3_Total	3.34573E-05	18485.9	0.999967
total	metal4		metal4_Total	1.29643E-05	7301.28	0.999987
total	metal5		metal5_Total	3.10500E-06	1703.79	0.999997
total	metal6		metal6_Total	7.86142E-07	426.453	0.999999
total	metal7		metal7_Total	2.92598E-09	1.49560	1.00000
total	metal8		metal8_Total	7.94702E-09	3.99032	1.00000
total	metal9		metal9_Total	1.67270E-09	1.17341	1.00000
total	metal10		metal10_Total	1.21001E-09	0.806671	1.00000
total	single.contact		single.contact_Total	0.000129304		0.999871
total	single.via		single.via_Total	8.29062E-05		0.999917

2. Select the **Chip Summary** tab in the Calibre RVE for DFM window and examine the CAA NDD scores for each Group in the Avg_Quality, via results, and WCA columns as shown in [Figure 5-7](#).

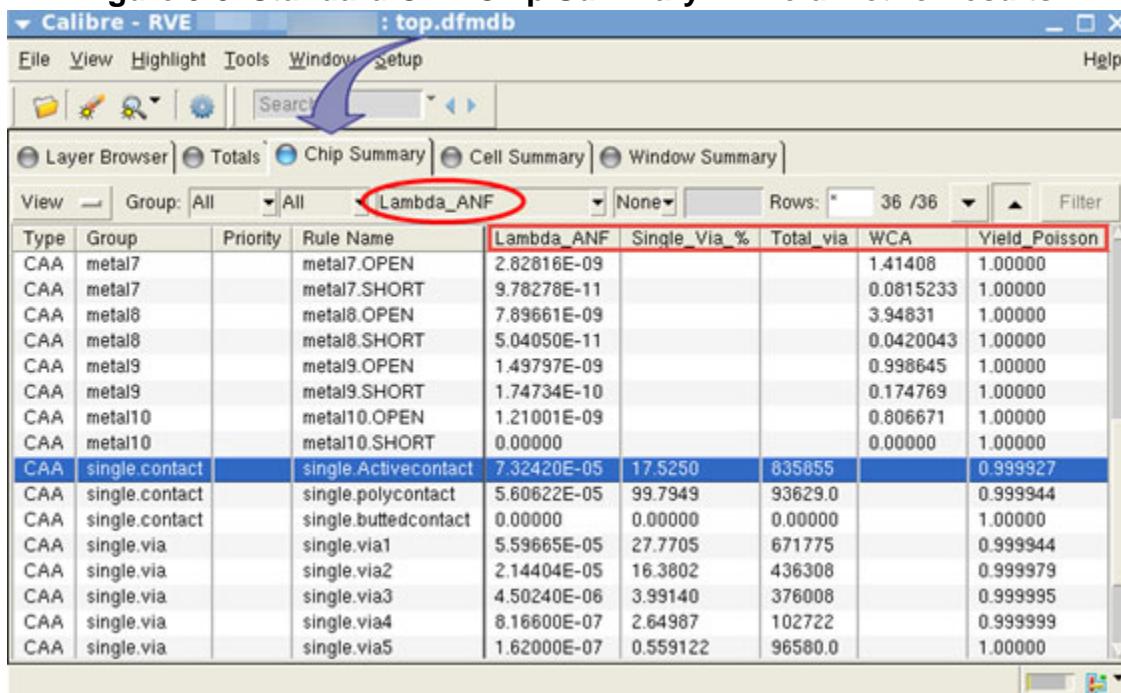
Figure 5-7. CAA NDD Chip Summary — Score Metric Results



Type	Group	Priority	Rule Name	Avg_Quality	Single_Via_%	Single_Via_Count	Total_via	WCA
CAA	metal4		metal4.OPEN	0.982319				5263.07
CAA	metal4		metal4.SHORT	0.985338				2047.84
CAA	metal5		metal5.OPEN	0.990292				1356.00
CAA	metal5		metal5.SHORT	0.997292				378.209
CAA	metal6		metal6.OPEN	0.997509				347.987
CAA	metal6		metal6.SHORT	0.999403				83.4534
CAA	metal7		metal7.OPEN	0.999988				1.63660
CAA	metal7		metal7.SHORT	0.999999				0.0615233
CAA	metal8		metal8.OPEN	0.999968				4.50809
CAA	metal8		metal8.SHORT	1.00000				0.0420043
CAA	metal9		metal9.OPEN	0.999991				1.30598
CAA	metal9		metal9.SHORT	0.999999				0.174769
CAA	metal10		metal10.OPEN	0.999993				1.01378
CAA	metal10		metal10.SHORT	1.00000				0.00000
CAA	single.contact		single.polycontact	0.00205065	99.7949	93437.0	93629.0	
CAA	single.contact		single.Activecontact	0.824750	17.5250	146484	835855	
CAA	single.contact		single.buttedcontact	1.00000	0.00000	0.00000	0.00000	
CAA	single.via		single.via1	0.754480	24.5520	164934	671775	
CAA	single.via		single.via2	0.865315	13.4685	58764.0	436308	
CAA	single.via		single.via3	0.960086	3.99140	15008.0	376008	
CAA	single.via		single.via4	0.973501	2.64987	2722.00	102722	
CAA	single.via		single.via5	0.994409	0.559122	540.000	96580.0	
CAA	single.via		single.via6	0.999929	0.0178078	17.0000	0.000070	

For comparison, examine the **Chip Summary** tab yield metrics and results from a standard CAA run as shown in [Figure 5-8](#).

Figure 5-8. Standard CAA Chip Summary — Yield Metric Results

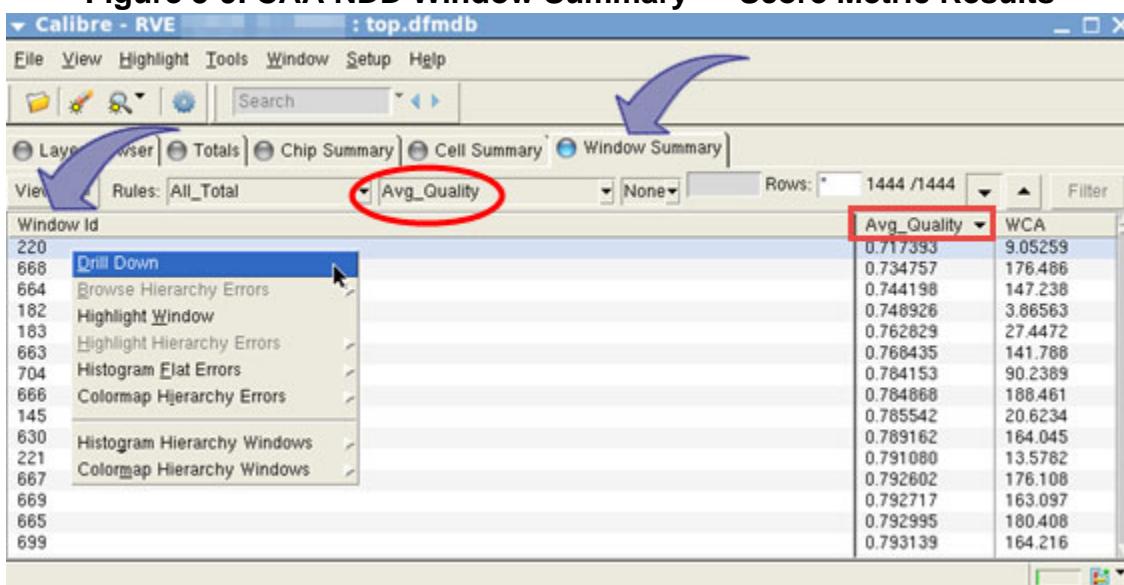


A screenshot of the Calibre RVE software interface. The window title is 'Calibre - RVE : top.dfmdb'. The menu bar includes File, View, Highlight, Tools, Window, Setup, and Help. The toolbar contains icons for file operations like Open, Save, and Print, along with a search function. The main area shows a table titled 'Chip Summary'. The columns are Type, Group, Priority, Rule Name, Lambda_ANF, Single_Via_%, Total_via, WCA, and Yield_Poisson. A red circle highlights the 'Lambda_ANF' column header. The data rows list various metal layers and their respective scores.

Type	Group	Priority	Rule Name	Lambda_ANF	Single_Via_%	Total_via	WCA	Yield_Poisson
CAA	metal7		metal7.OPEN	2.82816E-09			1.41408	1.00000
CAA	metal7		metal7.SHORT	9.78278E-11			0.0815233	1.00000
CAA	metal8		metal8.OPEN	7.89661E-09			3.94831	1.00000
CAA	metal8		metal8.SHORT	5.04050E-11			0.0420043	1.00000
CAA	metal9		metal9.OPEN	1.49797E-09			0.998645	1.00000
CAA	metal9		metal9.SHORT	1.74734E-10			0.174789	1.00000
CAA	metal10		metal10.OPEN	1.21001E-09			0.806671	1.00000
CAA	metal10		metal10.SHORT	0.00000			0.00000	1.00000
CAA	single.contact		single.Activecontact	7.32420E-05	17.5250	835855		0.999927
CAA	single.contact		single.polycontact	5.60622E-05	99.7949	93629.0		0.999944
CAA	single.contact		single.butteditcontact	0.00000	0.00000	0.00000		1.00000
CAA	single.via		single.via1	5.59665E-05	27.7705	671775		0.999944
CAA	single.via		single.via2	2.14404E-05	16.3802	436308		0.999979
CAA	single.via		single.via3	4.50240E-06	3.99140	376008		0.999995
CAA	single.via		single.via4	8.16600E-07	2.64987	102722		0.999999
CAA	single.via		single.via5	1.62000E-07	0.559122	96580.0		1.00000

3. Select the **Window Summary** tab in the Calibre RVE for DFM window to examine the CAA NDD scores by Window Id in the Avg_Quality results column.
- a. Right-click on a Window Id and select **Drill Down** as shown in [Figure 5-9](#).

Figure 5-9. CAA NDD Window Summary — Score Metric Results



A screenshot of the Calibre RVE software interface. The window title is 'Calibre - RVE : top.dfmdb'. The menu bar includes File, View, Highlight, Tools, Window, Setup, and Help. The toolbar contains icons for file operations like Open, Save, and Print, along with a search function. The main area shows a table titled 'Window Summary'. The columns are Window Id, Avg_Quality, and WCA. A red circle highlights the 'Avg_Quality' column header. The data rows list various window IDs and their corresponding scores. A context menu is open over the row with Window Id 220, showing options like 'Drill Down', 'Browse Hierarchy Errors', 'Highlight Window', etc.

Window Id	Avg_Quality	WCA
220	0.717393	9.05259
668	0.734757	176.466
664	0.744198	147.238
182	0.748926	3.86563
183	0.762829	27.4472
663	0.768435	141.788
704	0.784153	90.2389
666	0.784868	188.461
145	0.785542	20.6234
630	0.789162	164.045
221	0.791080	13.5782
667	0.792602	176.108
669	0.792717	163.097
665	0.792995	180.408
699	0.793139	164.216

In the **Drill Down** tab, examine the CAA NDD scores and via results by Group and Rule Name for the selected Window Id as shown in [Figure 5-10](#).

Figure 5-10. CAA NDD Window Summary Drill Down — Score Metric Results

Type	Group	Priority	Rule Name	Avg_Quality	Single_Via_%	Single_Via_Count	Total_via	WCA
CAA	metal9		metal9.OPEN	0.9998961				0.103892
total	metal9		metal9_Total	0.999409				0.118216
CAA	metal9		metal9.SHORT	0.999857				0.0143247
CAA	metal10		metal10.OPEN	0.999611				0.0388938
total	metal10		metal10_Total	0.999806				0.0388938
CAA	metal10		metal10.SHORT	1.00000				0.00000
CAA	single.contact		single.polycontact	0.00000	100.000	8.00000	8.00000	
total	single.contact		single.contact_Total	0.6386889				
CAA	single.contact		single.Activecontact	0.916667	8.33333	2.00000	24.0000	
CAA	single.contact		single.buttressedcontact	1.00000	0.00000	0.00000	0.00000	
CAA	single.via		single.via1	0.00000	100.000	7.00000	7.00000	
total	single.via		single.via_Total	0.00000				
CAA	single.via		single.via2	0.00000	100.000	1.00000	1.00000	
CAA	single.via		single.via3	0.00000	100.000	3.00000	3.00000	

4. Select the **Cell Summary** tab in the Calibre RVE for DFM window to examine cell information and CAA NDD scores by Cell Name in the Avg_Quality results column as shown in [Figure 5-11](#).

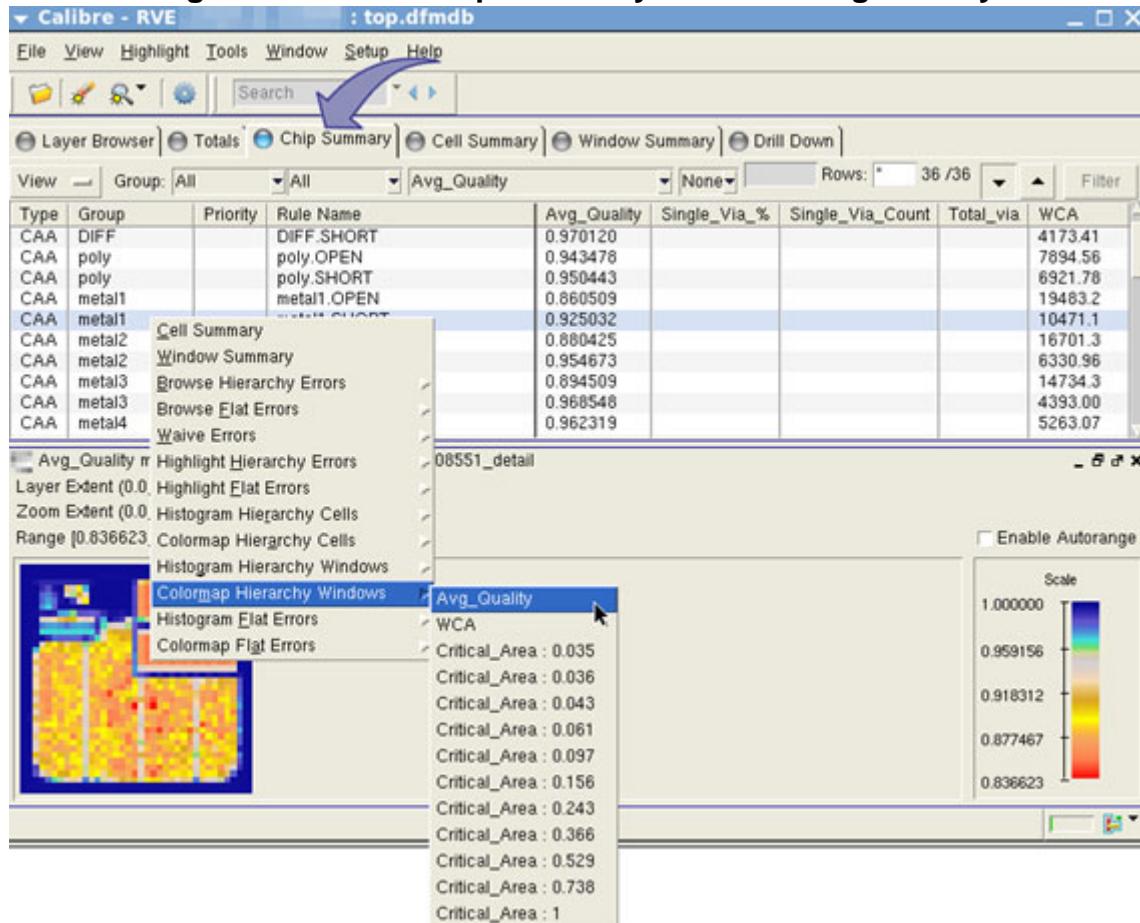
Figure 5-11. CAA NDD Cell Summary — Score Metric Results

Cell Name	Cell Area	Instance Count	Avg_Quality (H)	WCA (H)
NM	1.6311375	96	0.838895	1.30455
CN	1.725075	16	0.839200	1.36074
myram_word	23.92590625	768	0.840685	17.5938
myram_rdec	417.1013125	2	0.844713	434.339
myram	11558.7279	2	0.846559	15560.5
myram_block	1290.97978125	16	0.854373	1395.27
NA	1.1354875	6	0.865974	0.774477
AM1	0.81405	256	0.873700	0.445113
CA0	0.89409375	24	0.874332	0.245015
AM0	0.8110125	320	0.875100	0.402579
CA1	0.89409375	24	0.875447	0.209142
AI	4.2094125	12	0.893150	1.75973
adc_themo_2x	28.21185	15	0.893380	17.8983
BI	5.68978125	256	0.894115	3.80655
DFFNX1	12.40575	38	0.894403	6.64526

5. To view a CAA NDD colormap of Avg_Quality hierarchical window data for a rule, perform the following steps:
 - a. In the Calibre RVE for DFM window select the **Chip Summary** tab and left-click a Rule Name to select it.

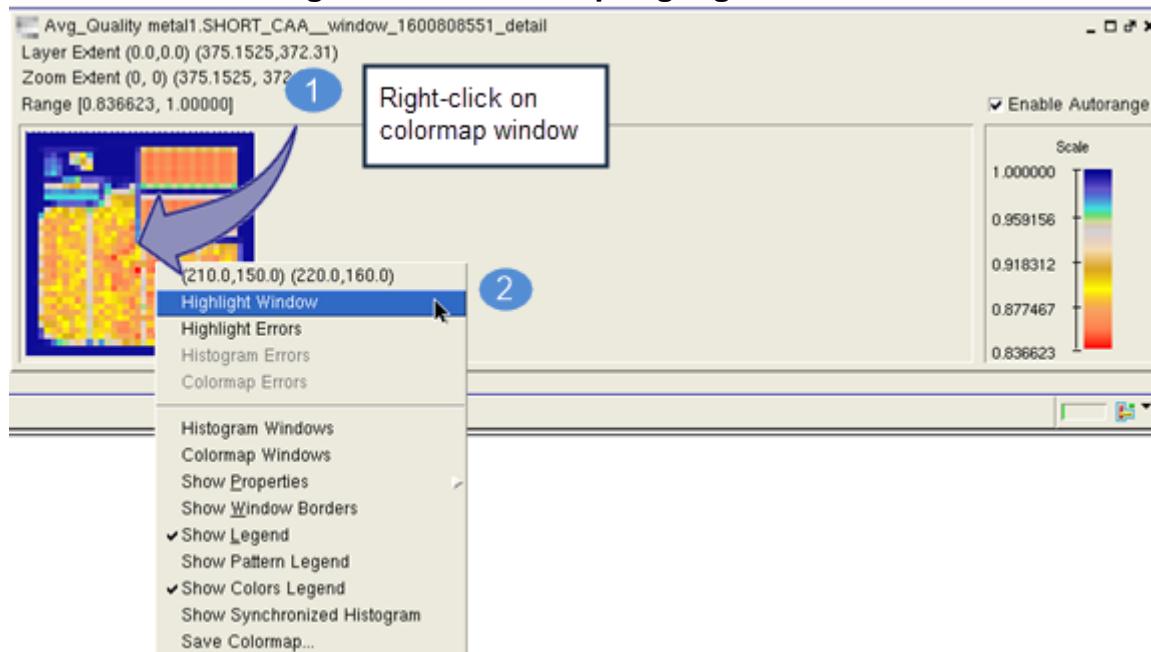
- b. Right-click the selected Rule Name and choose **Colormap Hierarchy Windows > Avg_Quality**.

Figure 5-12. Colormap Hierarchy Windows Avg_Quality



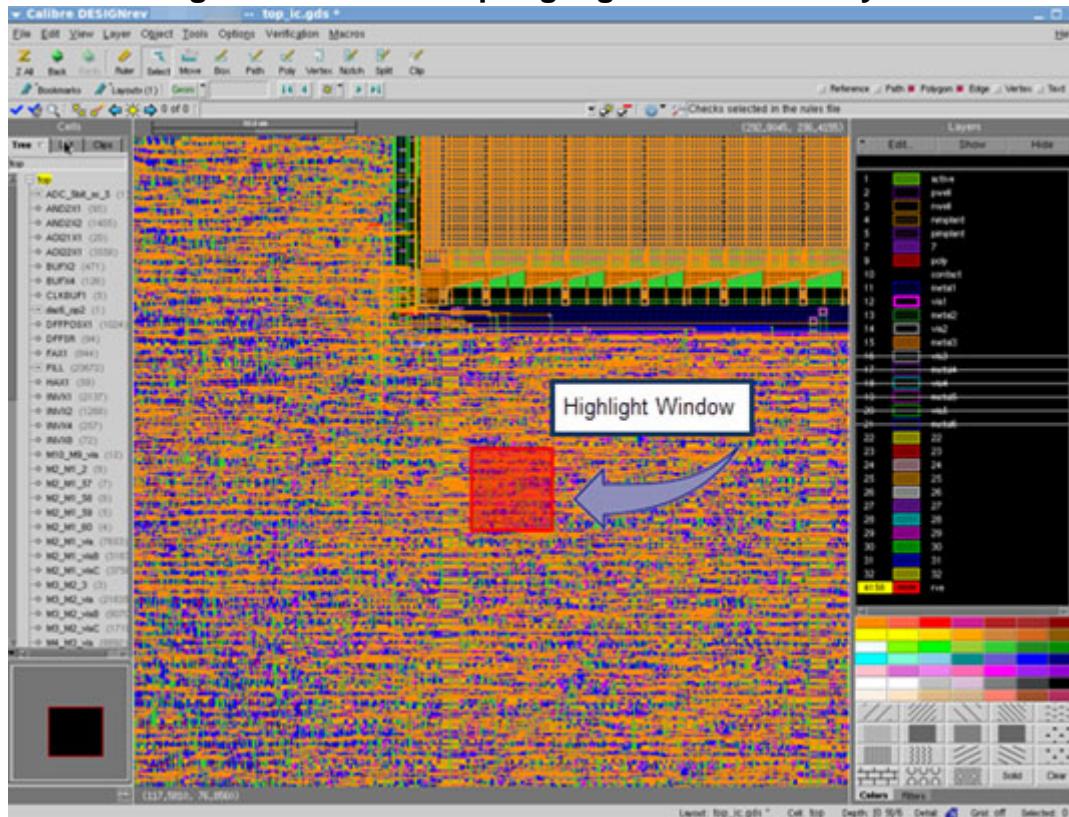
- c. Click **Enable Autorange** on the generated colormap Scale. Hover your cursor over the colormap and note that the red areas of the colormap have a lower Avg_Quality score than the blue areas. The highest Avg_Quality score is 1.000000.
- d. Right-click on a region (tile) of interest in the colormap window and select **Highlight Window**.

Figure 5-13. Colormap Highlight Window



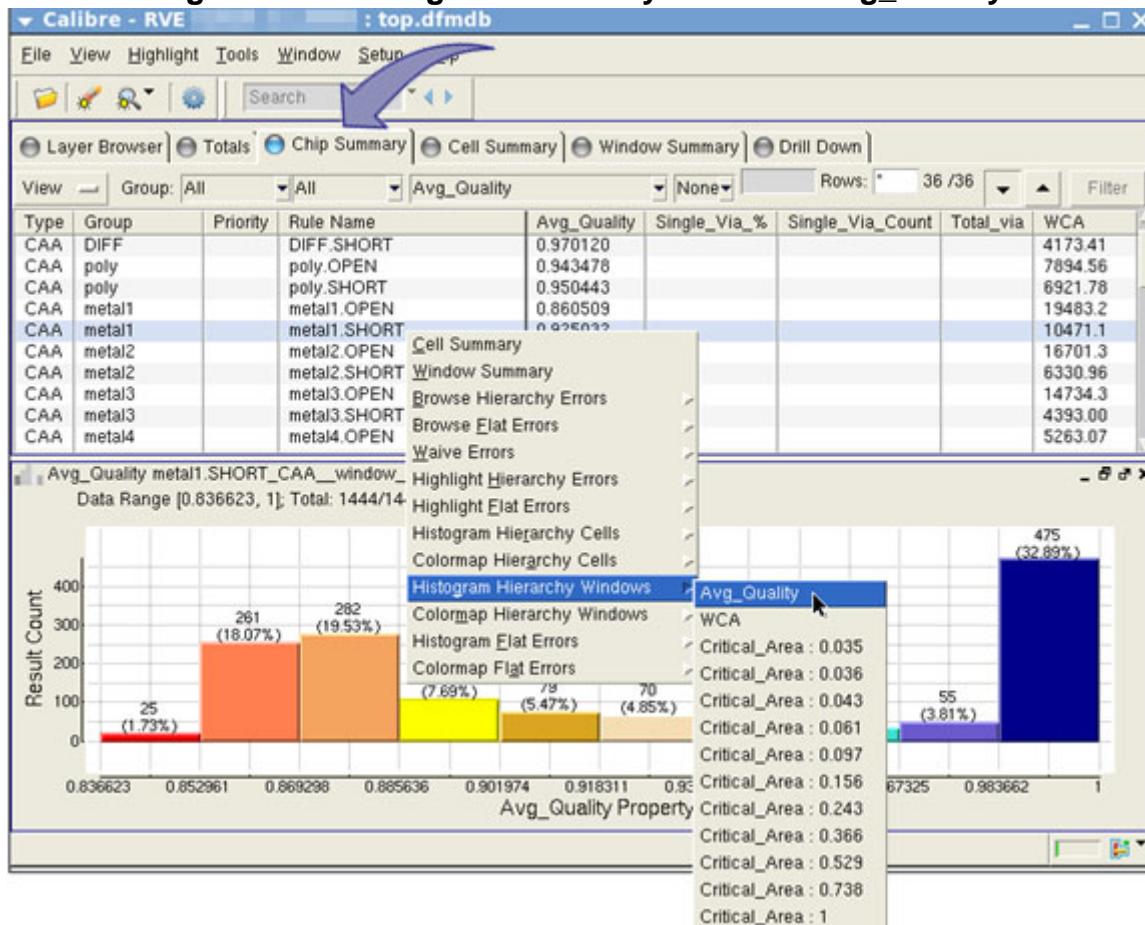
- e. View the highlighted region in a connected layout viewer.

Figure 5-14. Colormap Highlight Window in Layout



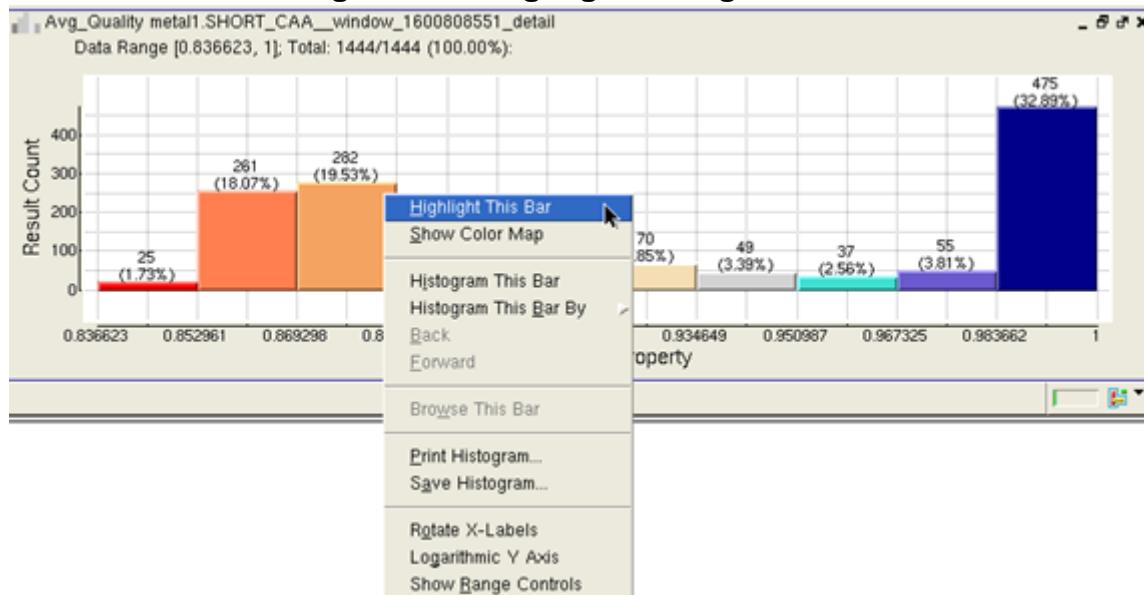
6. To view a histogram of Avg_Quality hierarchical window data for a rule, perform the following steps:
 - a. In the Calibre RVE for DFM window **Chip Summary** tab, left-click a Rule Name to select it.
 - b. Right-click the selected Rule Name and choose **Histogram Hierarchy Windows > Avg_Quality**.

Figure 5-15. Histogram Hierarchy Windows Avg_Quality



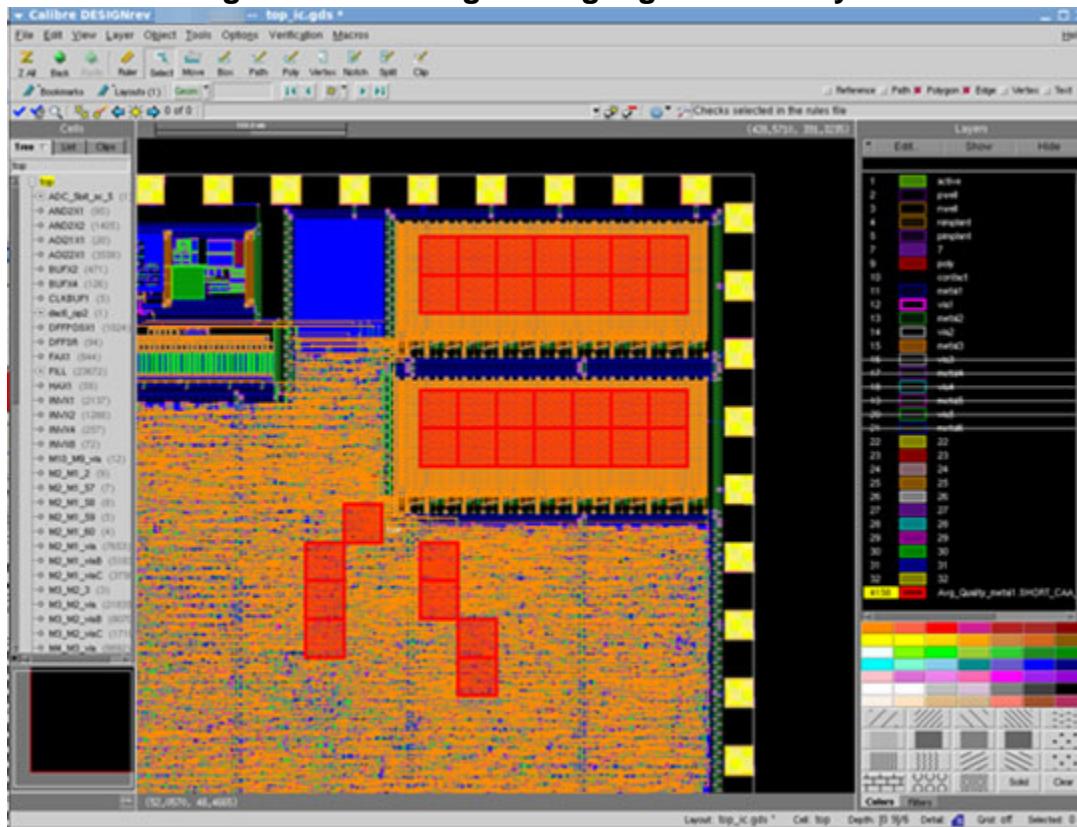
- c. Hover your cursor over the generated histogram to view the data range and count statistics. Note that a red color bar in the histogram has lower Avg_Quality Property scores than a blue color bar. The highest Avg_Quality Property score is 1.000000.
- d. Right-click on a histogram bar of interest and select **Highlight This Bar**.

Figure 5-16. Highlight Histogram Bar



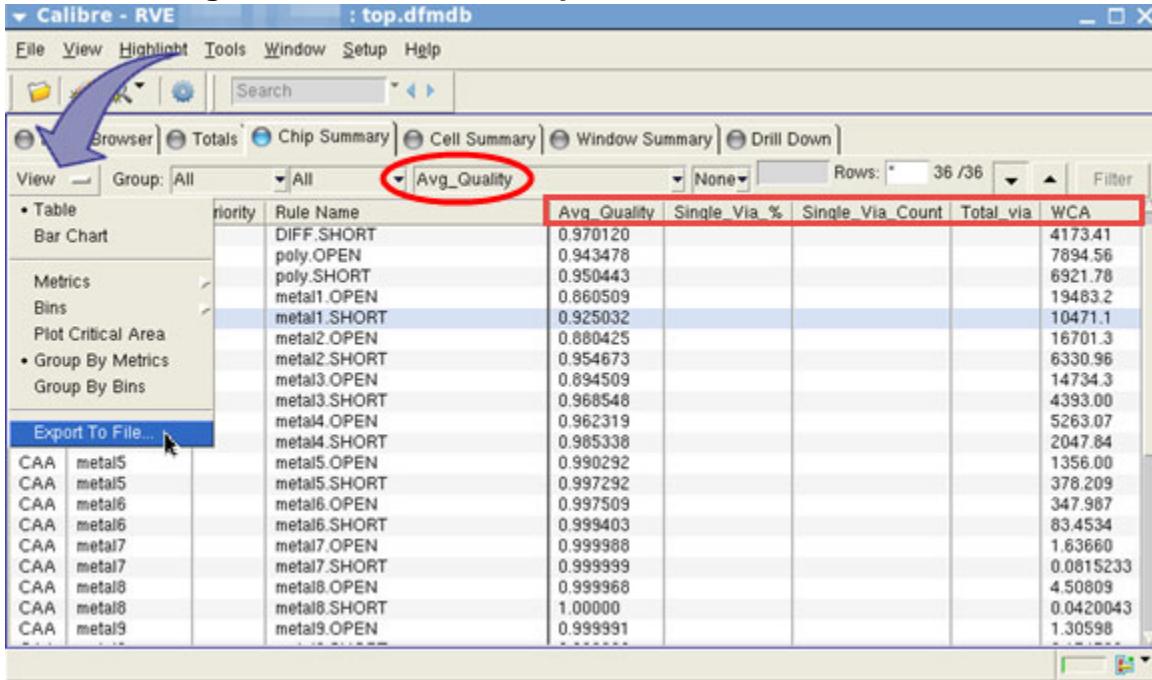
- e. View the highlighted region in a connected layout viewer.

Figure 5-17. Histogram Highlight Bar in Layout



7. To export Avg_Quality data to a report format file, perform the following steps:
 - a. In the Calibre RVE for DFM window **Chip Summary** tab, click the **View** menu and select **Export To File...** as shown in [Figure 5-18](#).

Figure 5-18. CAA NDD Export — Score Metric Results



The screenshot shows the Calibre RVE for DFM interface with the 'Chip Summary' tab selected. The 'View' menu is open, and the 'Export To File...' option is highlighted with a blue arrow. A red circle highlights the 'Avg_Quality' column header in the table. The table displays various rule names and their corresponding Avg_Quality scores, along with other metrics like Single_Via_% and Total_via.

Priority	Rule Name	Avg_Quality	Single_Via_%	Single_Via_Count	Total_via	WCA
Bar Chart	DIFF.SHORT	0.970120				4173.41
Metrics	poly.OPEN	0.943478				7894.56
Bins	poly.SHORT	0.950443				6921.78
Plot Critical Area	metal1.OPEN	0.860509				19483.2
• Group By Metrics	metal1.SHORT	0.925032				10471.1
Group By Bins	metal2.OPEN	0.880425				16701.3
	metal2.SHORT	0.954673				6330.96
	metal3.OPEN	0.894509				14734.3
	metal3.SHORT	0.968548				4393.00
	metal4.OPEN	0.962319				5263.07
	metal4.SHORT	0.985338				2047.84
CAA	metal5.OPEN	0.990292				1356.00
CAA	metal5.SHORT	0.997292				378.209
CAA	metal6.OPEN	0.997509				347.987
CAA	metal6.SHORT	0.999403				83.4534
CAA	metal7.OPEN	0.999988				1.63660
CAA	metal7.SHORT	0.999999				0.0815233
CAA	metal8.OPEN	0.999968				4.50809
CAA	metal8.SHORT	1.00000				0.0420043
CAA	metal9.OPEN	0.999991				1.30598

- b. Enter a filename for your report and click **OK**. The report is written to your current directory. By default, the report columns are sorted in ascending order. You can change the sorting order of a column by clicking the column header in the Calibre RVE for DFM window **Chip Summary** tab and re-running the report.
- c. View the report file Avg_Quality scores and via scores and counts. Note that the report shows only results for Avg_Quality metrics and suppresses information for Lambda or Yield metrics.

Chapter 6

Calibre DFM Explorer

Calibre DFM Explorer provides an interface for running several analysis tools in a simple flow.

You can use the tool to run lithography, pattern matching, and DFM scoring to analyze and refine your design.

DFM Explorer Tool Flow	131
DFM Explorer PDK Structure.....	133
DFM Explorer GUI Behavior.....	136
Using the DFM Explorer Tool	146

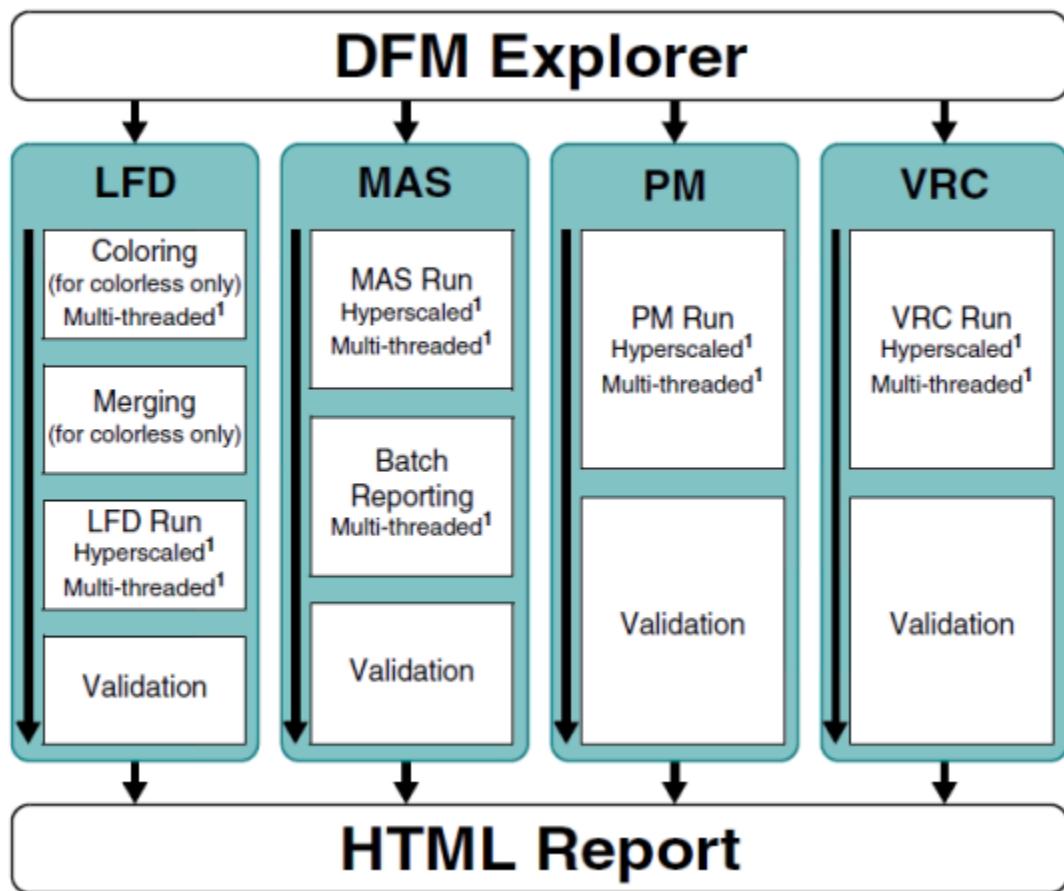
DFM Explorer Tool Flow

DFM Explorer (DE) provides a common cockpit for running several analysis tools.

Each analysis tool is part of the Calibre tool suite. DFM Explorer provides a convenient platform for setting up each tool. The analysis flow is shown in [Figure 6-1](#). All tools in the flow require foundry-provided technology information, including Calibre rule files.

- **Litho-Friendly Design (LFD)** — Calibre LFD finds hotspots in the design. You may select from regular LFD, Fast LFD (FLFD), or Incremental LFD (ILFD). Use ILFD for post-ECO analysis to save time.
- **Manufacturing Analysis and Scoring (MAS)** — A MAS run uses Calibre Critical Feature Analysis to identify and score sensitive features in the design.
- **Pattern Matching (PM)** — Calibre Pattern Matching identifies layout patterns in the design that have been identified as those which may potentially cause manufacturing defects.
- **Via Redundancy Check (VRC)** — The VRC analysis uses Calibre YieldAnalyzer features to score via redundancy in a layout.

Figure 6-1. DFM Explorer Tool Flow



Colored and Colorless Options for LFD and MAS

You can select from colored or colorless options for LFD and MAS analysis runs. A colored layout has been converted to a multi-patterned format. A colorless layout is the original drawn layout that has not been converted with multi-patterning.

In a MAS run, layout coloring is completed on the fly inside the MAS rule deck. The LFD colorless flow uses the DRC deck to perform a separate multi-patterning step. In the DFM Explorer GUI, this is indicated by a separate tab, **MPT LFD**.

Note

 It is not advisable to select colored and colorless in the same run. That is, do not select an LFD Colorless run and a MAS Colored run, or vice versa.

Environment Variable Settings for Different Calibre Versions

You can run each tool flow with a Calibre version that is different from the version used to run DFM Explorer. This capability is optional. It is useful when a specific version of Calibre is certified by the foundry for running a specific tool flow.

Set one or more of the environment variables shown in [Table 6-1](#) to specify a desired software version for Calibre. The examples in the table are presented for the c-shell format.

Table 6-1. DFME Environment Variables for Specifying Varied Calibre Versions

Environment Variable	Example Setting
MGC_HOME_FOR_LFD	<code>setenv MGC_HOME_FOR_LFD \ /tools/calibre/2015.4_16.11/lv_micro.aoi</code>
MGC_HOME_FOR_MAS	<code>setenv MGC_HOME_FOR_MAS \ /tools/calibre/2015.3_42.30/lv_micro.aoi</code>
MGC_HOME_FOR_PM	<code>setenv MGC_HOME_FOR_PM \ /tools/calibre/2015.1_26.16/lv_micro.aoi</code>
MGC_HOME_FOR_VRC	<code>setenv MGC_HOME_FOR_VRC \ /tools/calibre/2015.1_26.16/lv_micro.aoi</code>

The value of each variable must be a fully qualified pathname to a valid Calibre tree that can be used as \$MGC_HOME for that flow. Any tool flow (LFD, MAS, PM, or VRC) not specified by one of these variables uses the same version as that used to run DFM Explorer. If used, the variables must be set in your environment before starting DFM Explorer.

DFM Explorer PDK Structure

The process design kit (PDK) for DFM Explorer must be provided by the foundry. It must have a common directory structure for all tools in the flow (LFD, MAS, VRC, and PM). This ensures that technology information is properly found and loaded during a DE run.

File and Directory Naming Conventions

DFM Explorer automatically populates PDK paths and filenames for all selected analysis tools in the GUI if the following conventions are used for the filenames, where *tool_name* is LFD, MAS, PM, or VRC:

- *tool_name_MAIN.cal* — This is the main rule deck for the tool, for example MAS_MAIN.cal or LFD_MAIN.cal.
- *tool_name_review4DE.tcl* — This is a Tcl script. It is used in DE to generate a validation results summary.
- *tool_name.setvars* — This file contains shell script statements for setting environment variables. For MAS and LFD, the directory contains separate files for colored and colorless options. See also “[The .setvars File Format](#)”.
- *tool_name_report.cfg* — This file is only included in the MAS tool directory. It sets up the configuration for the HTML report generated for MAS results.

Additionally, the PDK files must be placed in a directory that follow this convention:

*tech_dir/*tool_name*/tool_name/4DE/filename*

where *tech_dir* is the path for the technology directory and *tool_name* is the analysis tool name (LFD, MAS, PM, or VRC).

The .setvars File Format

A .setvars file must be included for each tool in the foundry PDK. Two file formats are allowed:

- csh Format — DFM Explorer internally converts this format to bash syntax. Only the variable setting syntax is supported, as follows:

```
setenv VARIABLE_NAME VARIABLE_VALUE
```

For example, the csh format file may include:

```
setenv EXCLUDE_CELL "_dummy_"
setenv DRC_DIR $TECH_DIR
```

- bash Format — Full bash shell syntax is supported, including if-then blocks, for loops, and so forth. The first line of the file must be a statement that specifies a bash shell script:

```
#!/bin/bash
```

or,

```
#!/bin/sh
```

Other statements are included following the shell directive. For example:

```
#!/bin/bash

export EXCLUDE_CELL = "_dummy_"
export DRC_DIR = $TECHDIR

if [${PROCESS_METHOD} = "LFD" ] ; then
    export FN_RUN = 0
    export FC_RUN = 0
    ...

```

PDK Directory Example

For the following:

tool_name = LFD | MAS | VRC | PM

tech_dir = /mypath/PDK_1.0.0.2/S00_V1.0.0.2/DFM/4DE

the tool subdirectories are:

```
mypath/PDK_1.0.0.2/S00_V1.0.0.2/DFM/N10LP_CalibreLFD_V1.1.0.1/LFD/4DE
mypath/PDK_1.0.0.2/S00_V1.0.0.2/DFM/N10LP_CalibreMAS_V1.1.0.1/MAS/4DE
mypath/PDK_1.0.0.2/S00_V1.0.0.2/DFM/N10LP_CalibreVRC_V1.1.0.1/VRC/4DE
mypath/PDK_1.0.0.2/S00_V1.0.0.2/DFM/N10LP_CalibrePM_V1.1.0.1/PM/4DE
```

The strings “N10LP_Calibre” and “_V1.1.0.1” are part of the tool’s pathname and are identifiers for the PDK node and version. Node and version are not required, but the PDK must follow the directory structure template.

Inside the MAS subdirectory, a minimum set of files exists:

```
MAS_MAIN.cal
MAS_colored.setvars
MAS_colorless.setvars
MAS_report.cfg
MAS_review4DE.tcl
```

The dfme.ini File and the BEOL Stack List

It is possible to configure a predefined collection of the BEOL stack values with the dfme.ini file. When you start DFM Explorer, the tool searches for the file in the following locations:

- Predefined path set by the CALIBRE_DE_INIT_FILE environment variable.
- The current working directory, where DFME is started (\$CWD/dfme.ini).
- Your home directory (\$HOME/dfme.ini).

You must use the bash shell-command syntax. To configure the BEOL_STACK_LIST, include the following:

```
export BEOL_STACK_LIST = <beol stack elements>
```

For example,

```
export BEOL_STACK_LIST = "9M_3Mx_4Dx_1Gx_1Iz_LB 11M_4Mx_5Dx_1Gx_1Iz_LB
11M_3Mx_6Dx_1Gx_1Iz_LB 11M_4Mx_5Dx_1Gx_1Iz_LB"
```

Note

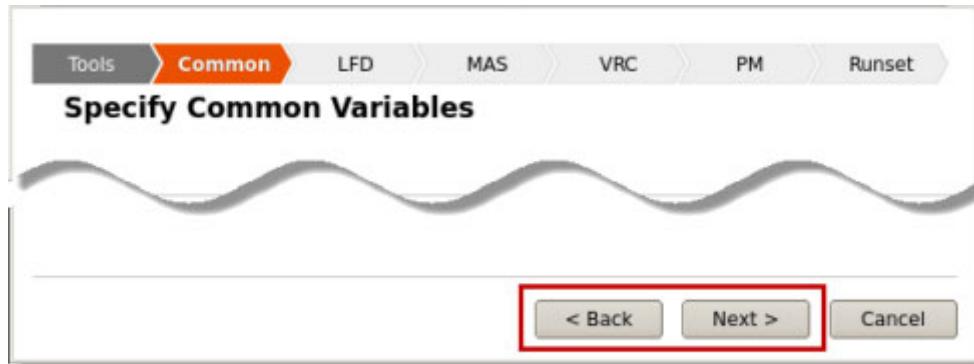
 You may also use the dfme.init file to define other default values. Use the bash shell-command syntax when doing so.

DFM Explorer GUI Behavior

The DFM Explorer GUI presents several tabs, one for each of the analysis tools and others for setting up and reviewing run information. The behavior of each tab is similar, with a few exceptions.

GUI Navigation

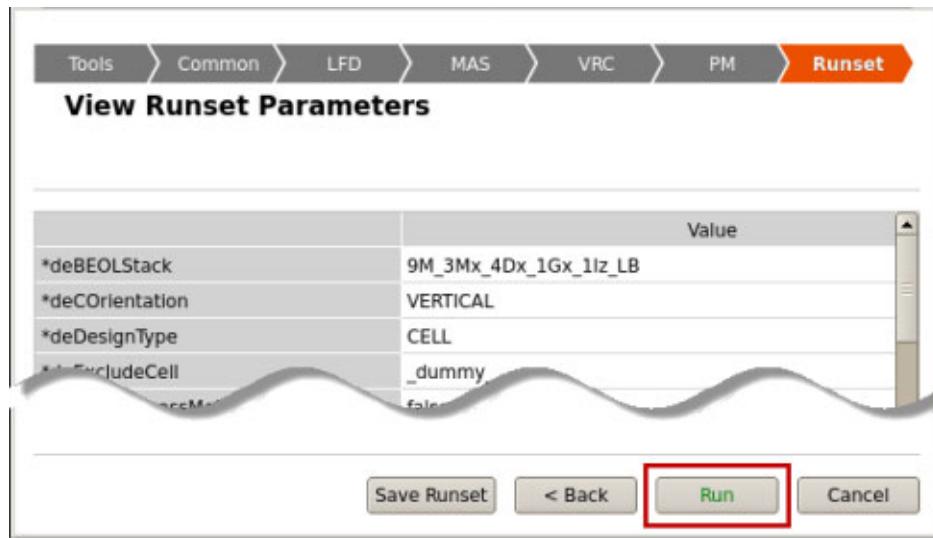
You can navigate through the **Common**, **LFD**, **MAS**, **VRC**, and **PM** tabs by using the **Back** and **Next** buttons on the bottom of each tab.



In the **Tools** tab you can click the **Load Runset** button to load a previously saved runset file. The runset file is saved in the output directory and overwritten with every new run.



In the **Runset** tab you can click the **Save Runset** button to save your runset for later use. The runset file only contains the fields that have been filled or checked throughout the DFME GUI.



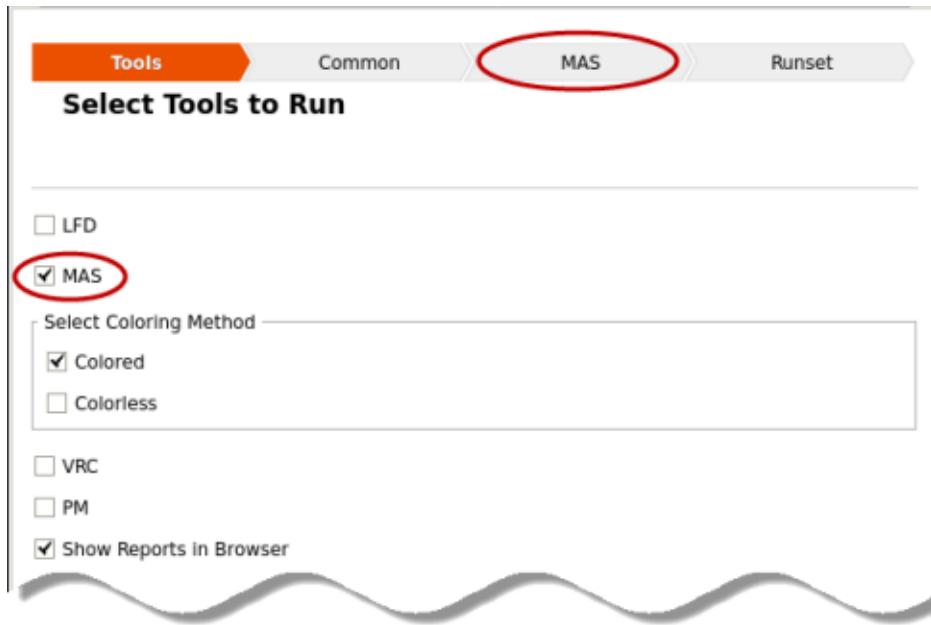
GUI Text Fields and Buttons

Certain entered fields and buttons must be green before clicking the **Run** button in the **Runset** tab. Buttons and fields change from red to green when you have specified all required information correctly.

Analysis Tool Selection

You must select at least one analysis tool in the **Tools** tab for a valid run. For each tool you select, a tab appears in the DE GUI.

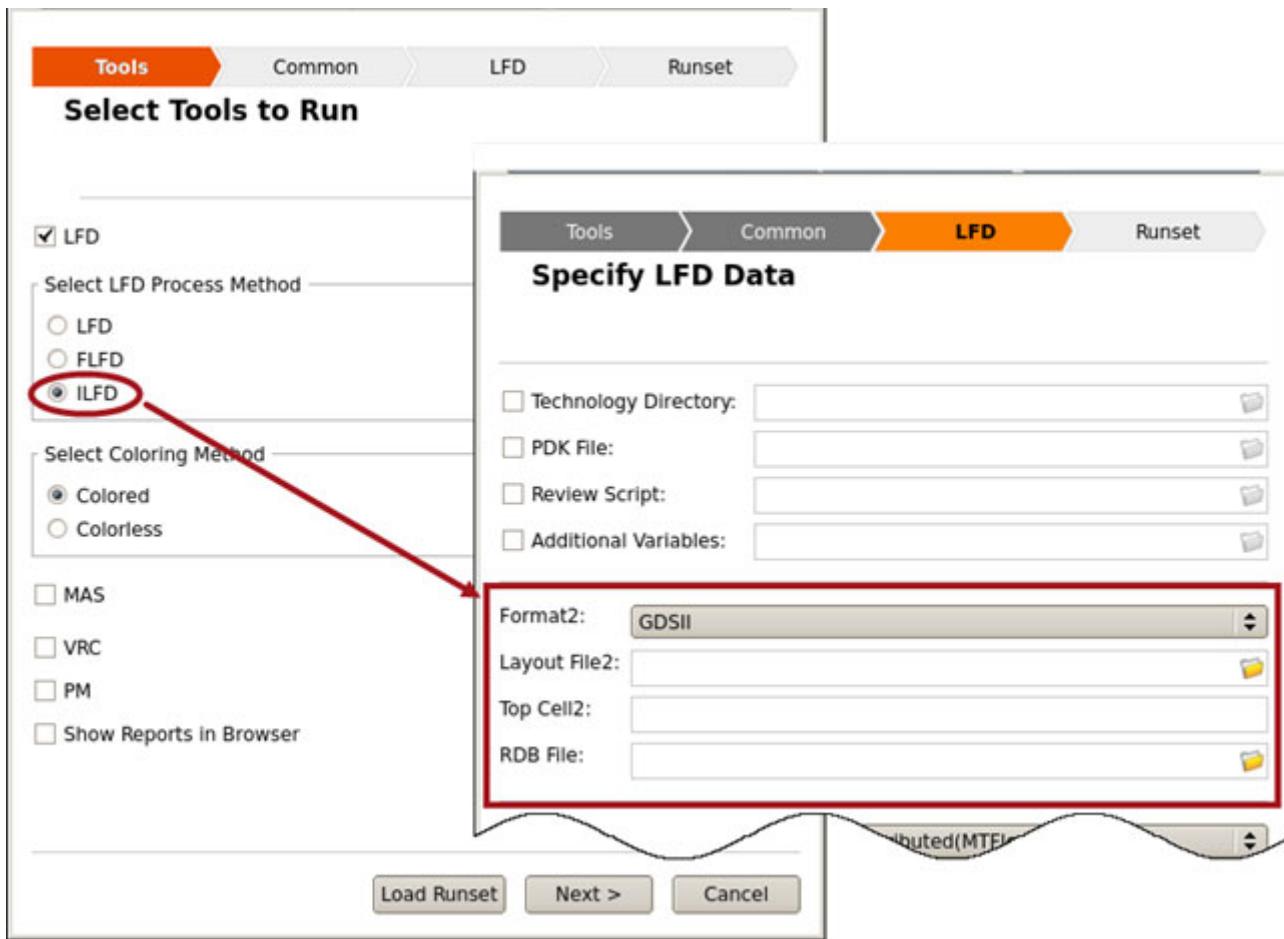
If you do not select an analysis tool, the **Run** button in the **Runset** tab remains red.



Incremental LFD (ILFD) Run Information

When you select **ILFD** in the **Tools** tab, the **LFD** tab is configured to include run information for a second layout.

The post-ECO layout information is entered in the **Common** tab. The pre-ECO layout information is entered in the **LFD** tab by specifying Layout File2, Top Cell2, and RDB File. The RDB File locates the hotspot information that the ECO fixed.

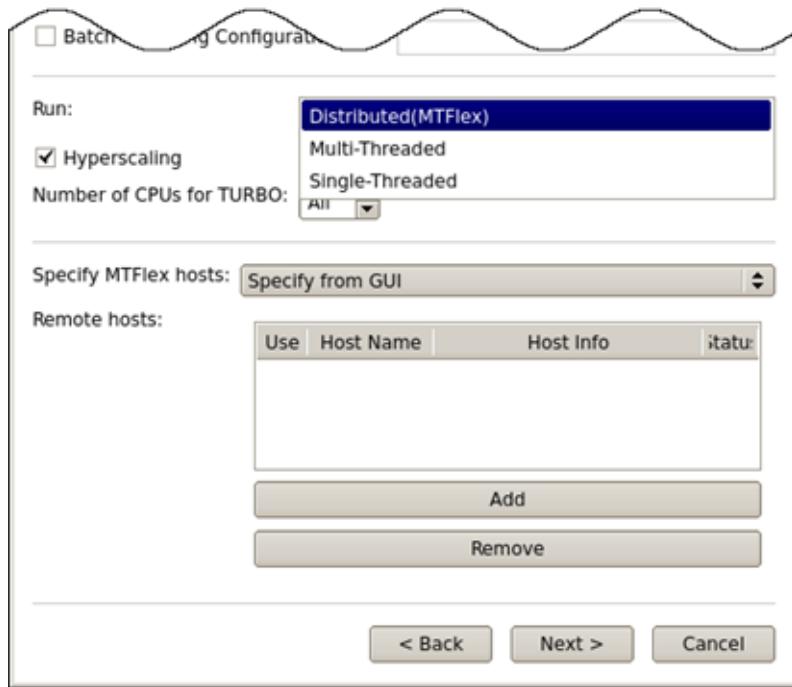


Runset Information Review

Be sure to review the runset parameters presented in the **Runset** tab. If there is incomplete or missing information, the **Run** button remains red.

Single-Threaded, Multi-Threaded, and Distributed (MTflex) Run Options

You can configure single-threaded, multi-threaded, or distributed (MTflex) run options from each tool tab. The only exception to this is for the LFD Colorless option. For details, see “[Colored and Colorless Selection for LFD and MAS](#)”.



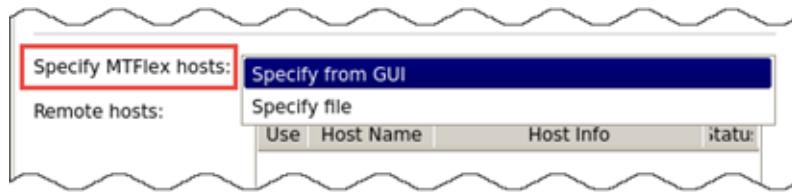
Hyperscaling and CPUs for the -turbo mode are valid for distributed and multi-threaded runs only:

- Check or uncheck **Hyperscaling**. The default setting is checked.
- If you know how many CPUs you have, specify this with **Number of CPUs for TURBO**. The default is **All** CPUs.

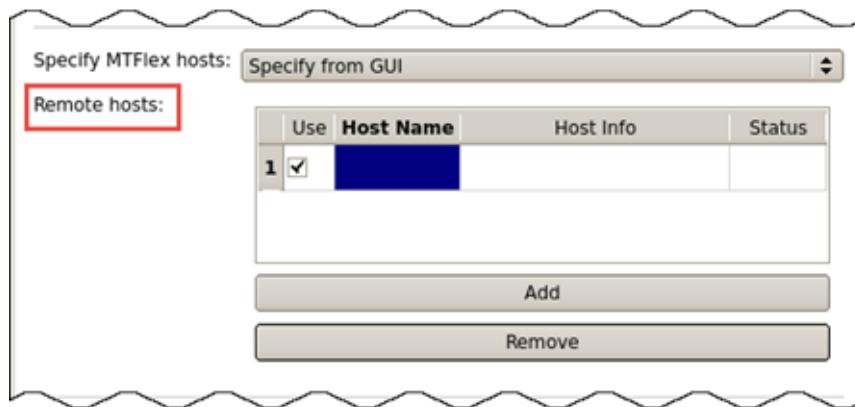
Distributed (MTflex)

This is the default option. The information you must provide for a Distributed run depends on the selected **Run on** option (LMP or LSF) in the **Common** tab.

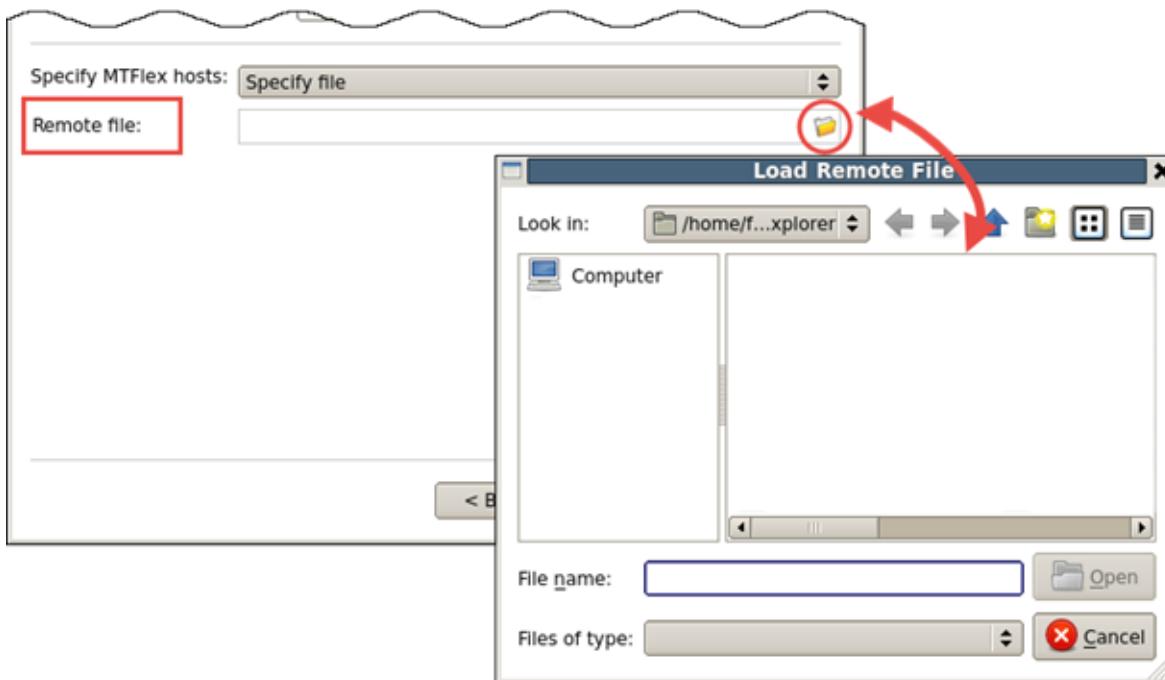
When you select **LMP**, you must specify remote hosts from the GUI or from a configuration file. You can do this from the tool tab (LFD, MPT LFD, MAS, VRC, or PM).



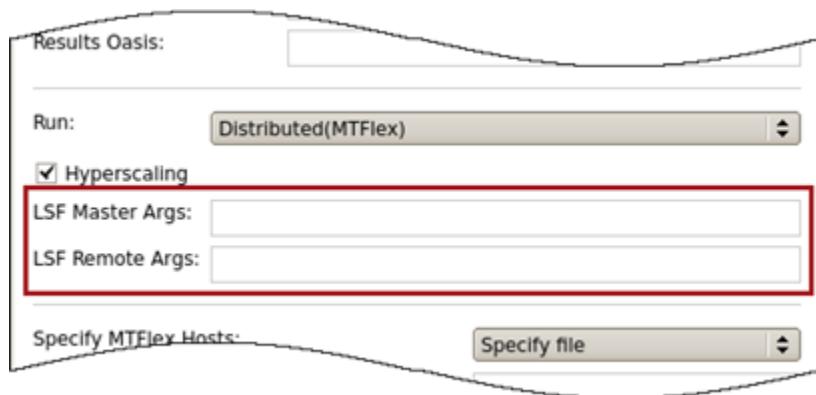
- **Specify from GUI** — This is the default option. Specify the desired hosts in the **Remote hosts** table. Use the **Add** and **Remove** buttons to create a list of hosts for your run.



- **Specify File** — You must provide a filename for the remote host configuration file. You can enter the location or browse to it by clicking the open file icon.



When you select **LSF**, you must also specify remote hosts from the GUI or from a configuration file, in addition to additional LSF arguments. You can do this from the tool tab (LFD, MPT LFD, MAS, VRC, or PM).



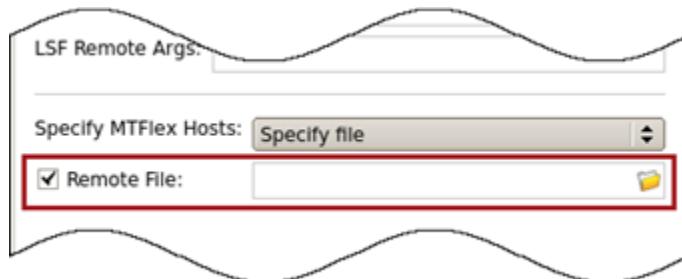
Note

LSF Master Args is available for Single-Threaded, Multi-Threaded, and Distributed (MTflex) modes. **LSF Remote Args** is only available for the Distributed (MTflex) mode.

If you select **Specify from GUI** for **Specify MTflex Hosts**, then use the **Remote Hosts** field to add or remove hosts.

If you select **Specify file**, there are two options to provide the remote machine information:

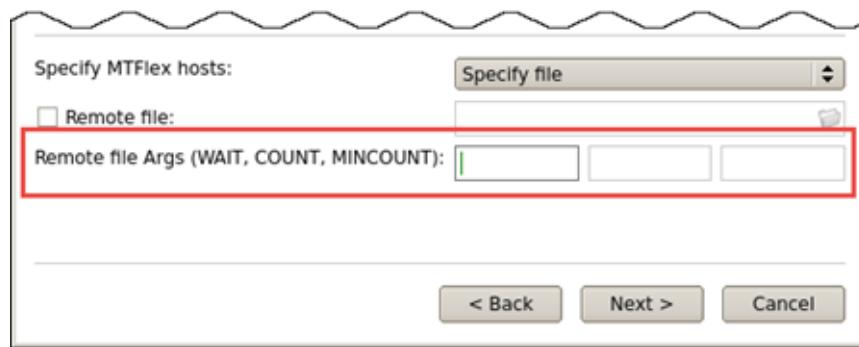
- If you select **Remote File**, enter or browse to the filename and location for the file.



- If you do not select **Remote File**, you must provide parameters that are used to select remote hosts. These are:
 - a. WAIT — Specifies a wait time (in seconds) which the local host waits for making a connection to a remote host.
 - b. COUNT — Specifies the number of remote hosts that the local host connects to.
 - c. MINCOUNT — Specifies the minimum number of remote hosts that must be connected before the WAIT time expires.

Note

 WAIT, COUNT, and MINCOUNT only apply to an LSF run.



DFM Explorer generates a configuration file which contains remote host information before executing the analysis run. These parameters and the generated configuration file are used with the LSF run method.

For more detailed information, see “[Calibre MTflex Processing](#)” and “[Creating a Configuration File](#)” in the in the *Calibre Administrator’s Guide*.

Note

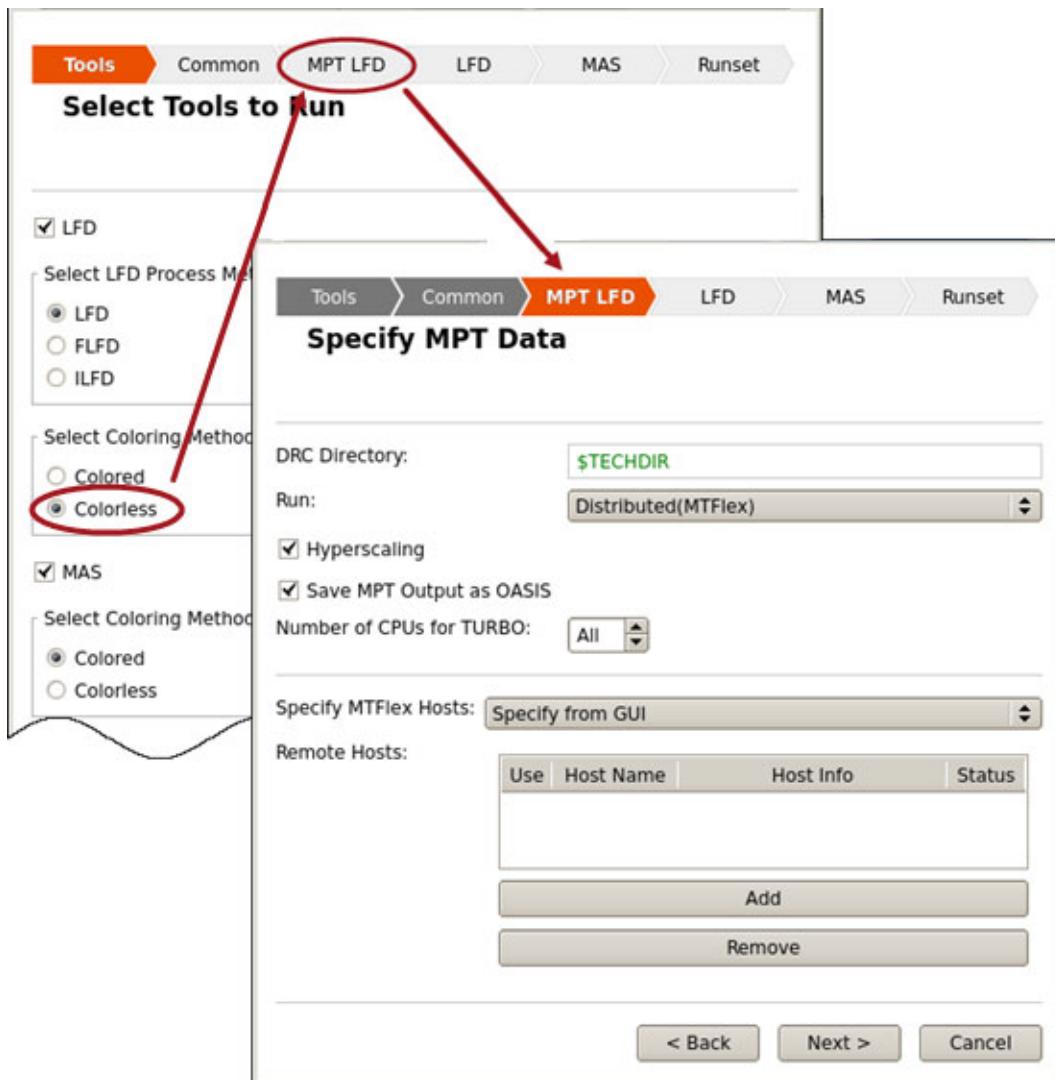
By default, DFM Explorer generates log files when using LSF. These files are found in *output_dir/toolname/lsf_logs*, where *output_dir* is the path to your generated output and *toolname* is the name of the analysis tool.

Multi-Threaded and Single-Threaded

Multi-threaded allows hyperscaling and the turbo run option. You can specify the number of CPUs for a multi-threaded run. Single-threaded will execute the run on your local host with a single CPU.

Colored and Colorless Selection for LFD and MAS

You can select from **Colored** and **Colorless** analysis options for LFD and MAS runs. When you select **Colorless** for **LFD**, an additional tab appears.



The **MPT LFD** (Multi-Patterning LFD) tab provides inputs for a DRC rule file directory, MPT OASIS output file selection, and setting up distributed, multi-threaded, and single-threaded run modes. PDK information is not included in this tab.

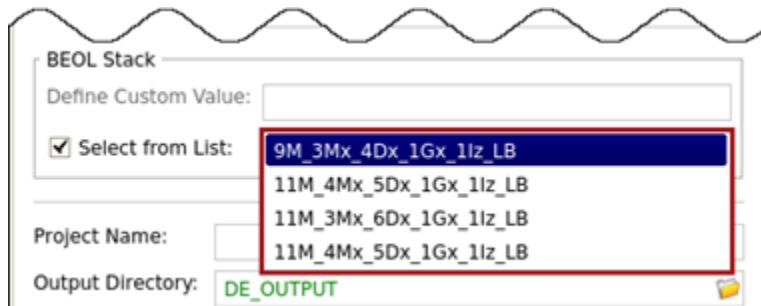
Use the **DRC Directory** field to enter a specific rule file to use during an MPT LFD run. The default location is the technology directory (\$TECHDIR).

The **Save MPT output as OASIS** option appears only if the layout system is not set to OASIS. Select this option to save the MPT output in OASIS format.

For more detailed information on multi-processor runs, see “[Single-Threaded, Multi-Threaded, and Distributed \(MTflex\) Run Options](#)”.

BEOL Stack Settings

You can define the BEOL (back end of line) stack in one of two ways: custom definition through the GUI or reading the dfme.ini file. When you create a dfme.ini file, the BEOL stack list is pre-configured in the GUI and the **Select from List** option is enabled. To specify a custom value, uncheck **Select from List** and use Define Custom Value to provide a value.



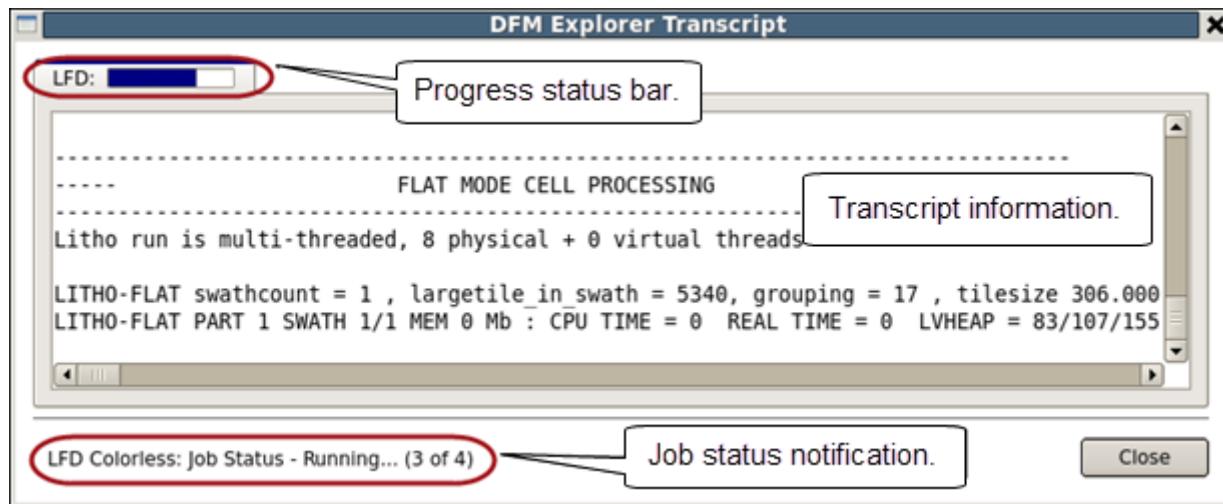
For more information about the dfme.ini file, see “[The dfme.ini File and the BEOL Stack List](#)” on page 135.

Transcript Window

After starting a DFME run, job status information is displayed in the DFM Explorer Transcript window. A short description of the current job stage is provided at the bottom of the window and a progress bar shows the progress of the run.

If a run fails, the progress bar turns red, indicating that the flow did not run to completion.

You can close this window at any time without affecting the DFME run.



Error Handling

For any flow (MAS, LFD, and so forth), errors may occur which stop the progress of the tool. Only fatal errors cause the flow to fail. Harmless errors are generally ignored and do not prohibit DFME from completing a run.

Using the DFM Explorer Tool

You can start DFM Explorer with or without a runset file.

Prerequisites

- The following licenses are required:
 - Calibre YieldAnalyzer licenses for the DFM Explorer GUI, MAS, and VRC.
 - Calibre nmDRC and nmDRC-H licenses.
 - Calibre LFD license.
 - Calibre Pattern Matching license.
- Valid PDK directory structure and PDK files for all tools selected in a run.

Procedure

1. Start the DFM Explorer tool.
 - a. To start the tool normally, enter the following at a shell prompt:

```
dfmexplorer
```

- b. To start the tool with an existing sunset file, enter the following at a shell prompt:

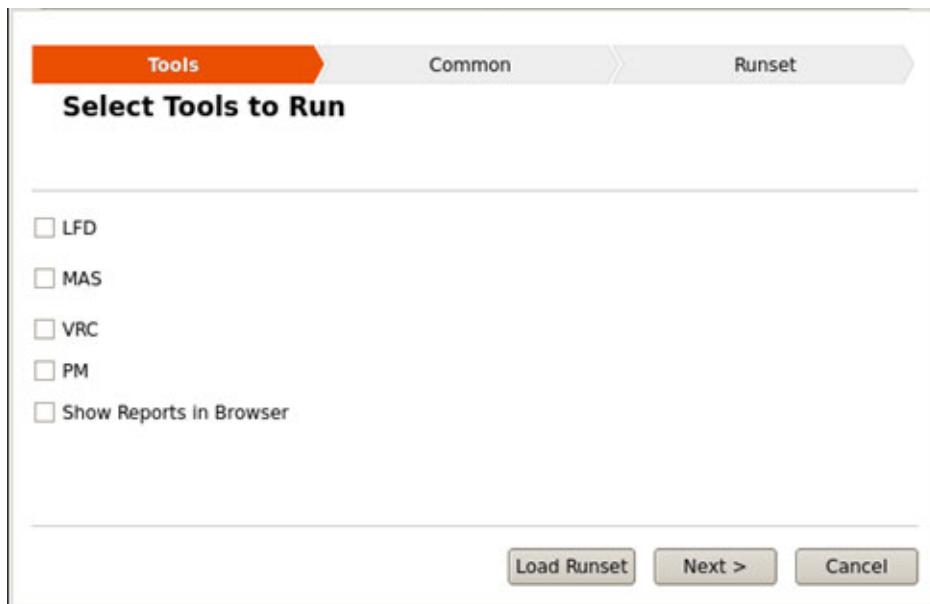
```
dfmexplorer -runset <filename.runset>
```

where *filename.runset* was created with DFM Explorer.

Tip

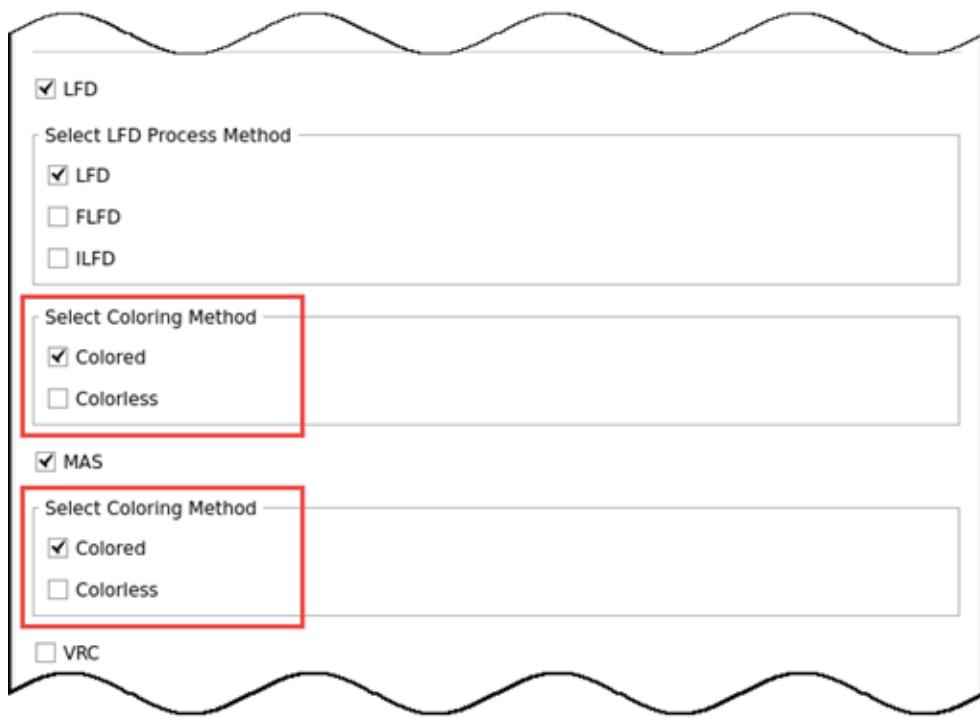
i You can set an environment variable so that a standard sunset file is always loaded in the tool. Set the MGC_CALIBRE_DE_RUNSET_FILE variable to a valid sunset file (include the pathname, as necessary). This file is automatically loaded if DE is invoked without the -runset command line option.

2. Select the analysis tools from the **Tools** tab.



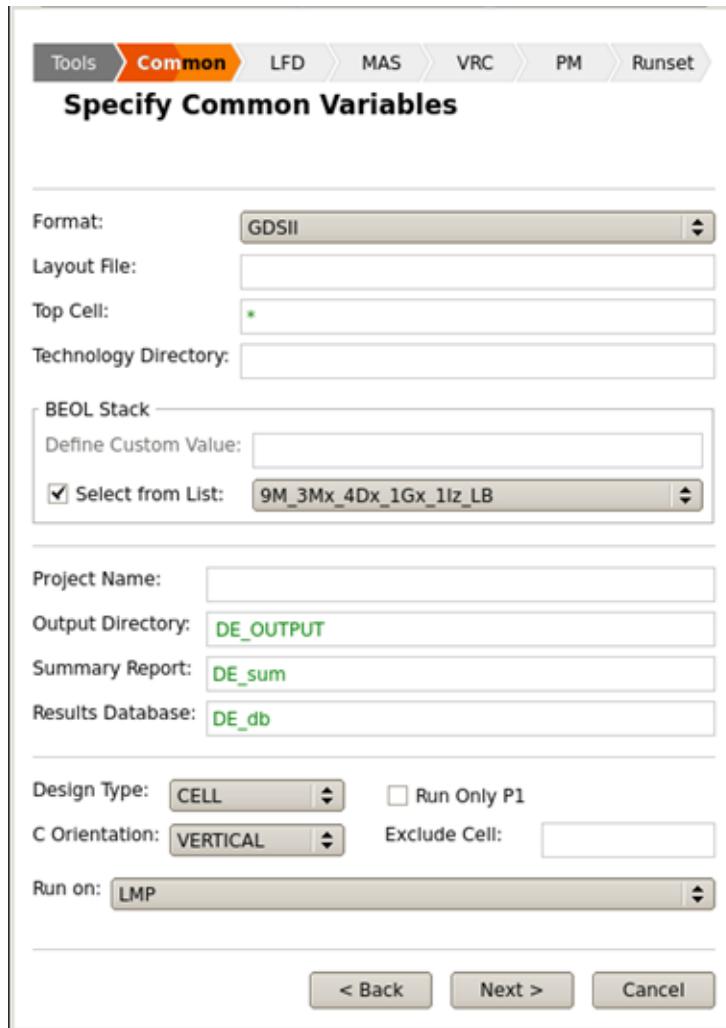
- a. Select from **LFD**, **MAS**, **VRC**, **PM**, or any combination of these tools.

Selecting **MAS** or **LFD** changes the **Tools** tab to include options for **Colored** and **Colorless** runs. The default setting is **Colored** for both tools.



- b. Optional — Select the LFD process method. You can select **LFD** (regular LFD), **FLFD** (fast LFD), or **ILFD** (incremental LFD).
- c. Click **Show Reports in Browser** to enable the run report to appear in your default browser. The report automatically refreshes as each tool runs.
- d. Optional — Click **Load Runset** to load an existing runset saved from a previous run.
- e. After selecting the desired tools, or loading a runset, click **Next**.

3. Enter shared tool setup information in the **Common** tab.



- a. Enter the layout information.
 - **Format** — Select the layout format: GDSII or OASIS.
 - **Layout File** — Click the file folder icon to browse to the file location or type in the filename (include the path as necessary).
 - **Top Cell** — Enter the layout's top cell name. You can also use an asterisk (*) in place of the top cell name.
 - **BEOL Stack** — Enter the back-end of line (BEOL) process stack string. The BEOL string includes the metal layers used in the process technology. Allowed values for the BEOL stack are provided by the foundry.

For more information on specifying the BEOL stack, see “[The dfme.ini File and the BEOL Stack List](#)” on page 135 and “[BEOL Stack Settings](#)” on page 145.

Note

 The BEOL string is not checked by the tool. The meaning of the string is not documented except by the foundry that provides the process technology. This value is typically included in the foundry's technology Design Rule Manual (DRM).

- **Technology Directory** — Enter the common technology directory path.

Note

 Do not enter any path that can be interpreted as the current working directory for **Technology Directory**; this includes entering “.” or “./”. This avoids possible naming conflicts between the output directory and PDK directory names. If this occurs, the **Run** button in the **Runset** tab turns red, indicating a naming conflict.

- b. Enter output-related information.

The output directory path for a DE run is created as follows:

output_dir/toolname/project_name/rdb/results_db_toolname

The directory path is composed of the following:

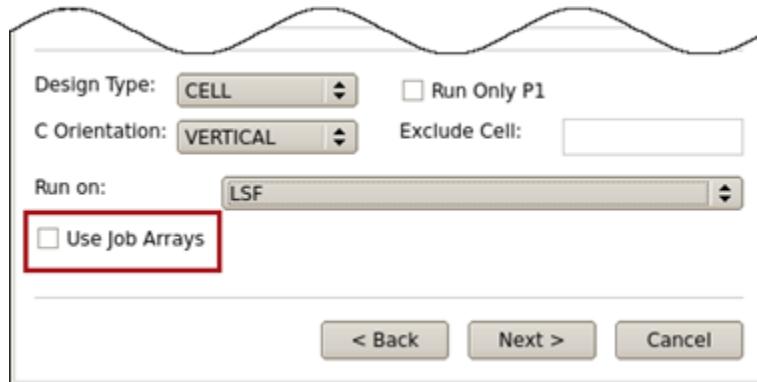
- **Output Directory** (*output_dir*) — Enter a name for your output directory. The default name is DE_OUTPUT. This directory includes the output for each tool you select to run. Output includes HTML-format reports, run data, shell scripts, and output databases.
- *toolname* — This is provided by DE: MAS, LFD, VRC, or PM.
- **Project Name** (*project_name*) — Enter a project name for your DE run. This is used as a location for results under each tool directory in the main output directory. This field does not have a default value.
- **Results Database** (*results_db*) — This is a required field. The default value is DE_db. Optionally, you can enter a name for the Calibre output database. The *toolname* is concatenated to the end of *results_db* (DE_db_toolname).
- **Summary Report** — This is a required field. The default value is DE_sum. Optionally, you can enter a name for the tool summary report file. This file is stored in *results_db_toolname* directory.

- c. Enter run-related information.

- **Design Type** — Select from CELL, CHIP, or STD (standard cells).
- **C Orientation** — Select the critical orientation: VERTICAL or HORIZONTAL. Refer to your foundry design rule manual for this information.
- **Run Only P1** — Check this option to run only Priority 1 LFD rules.

- **Exclude Cell** — Specify any cells names in the design to be excluded from the analysis run. This is a required field without a default value. Specify a dummy cell name if there are no cells to be excluded.
- **Run on** — Select the run method: LMP (local multi-processor) or LSF (load sharing facility).

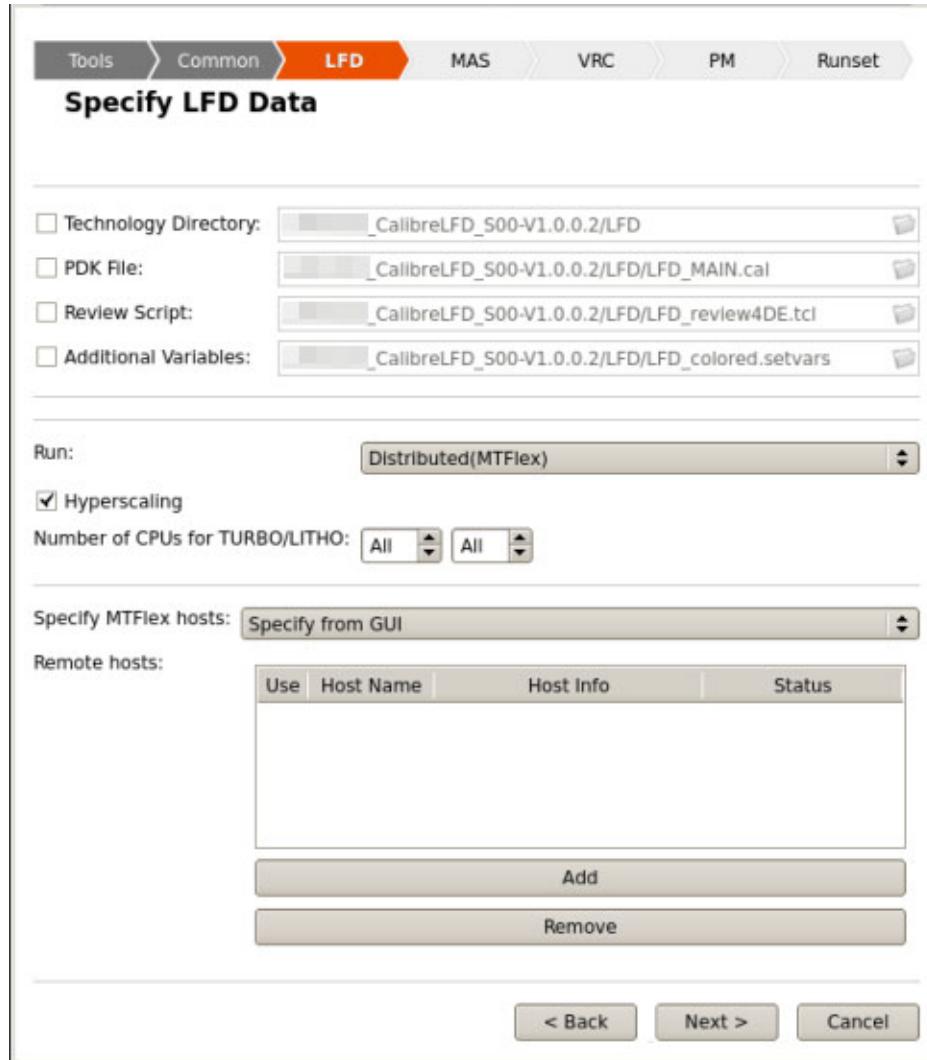
If you select LSF, the **Use Job Arrays** checkbox appears. Use this option to select whether or not to use job arrays. Selecting the checkbox directs DFME to include generated job arrays in the *_mtflex setup files.



For more information about selecting LMP or LSF for the **Run on** option, see “[Single-Threaded, Multi-Threaded, and Distributed \(MTflex\) Run Options](#)” on page 140.

- d. Click **Next** to proceed.
4. Enter and review the setup information for each tool you selected in the **Tools** tab.

With minor exceptions, all tool tabs require the same information. You can navigate through the tool tabs by clicking **Back** and **Next**.



- a. Review or modify the technology and PDK information.

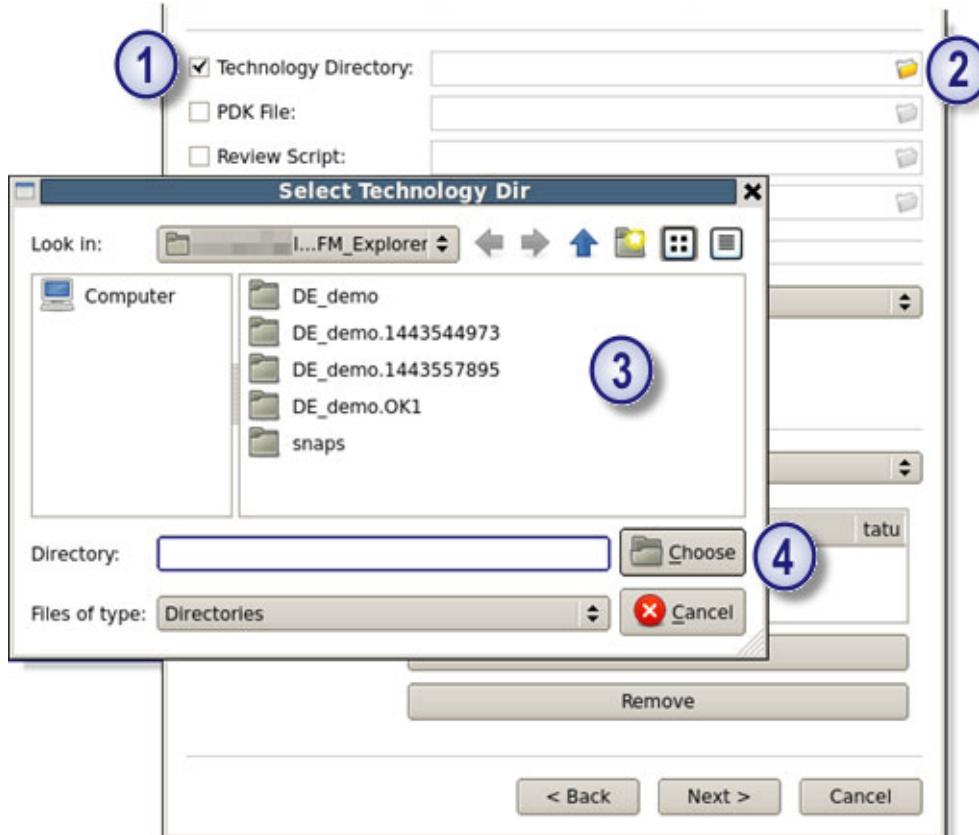
Tip

i DFM Explorer uses the **Technology Directory** path in the **Common** tab to look for the tool-related PDK files. If your PDK's directory structure and file-naming convention follow the formats described in “[DFM Explorer PDK Structure](#)” on page 133, then these fields are automatically filled.

To manually enter a file or directory location in the tool’s setup tab:

- i. Click checkbox next to the desired field (1).

- ii. Click the open file folder icon (2). This opens a file browsing window (3).
- iii. Navigate to the desired location or file and click **Choose** (4).

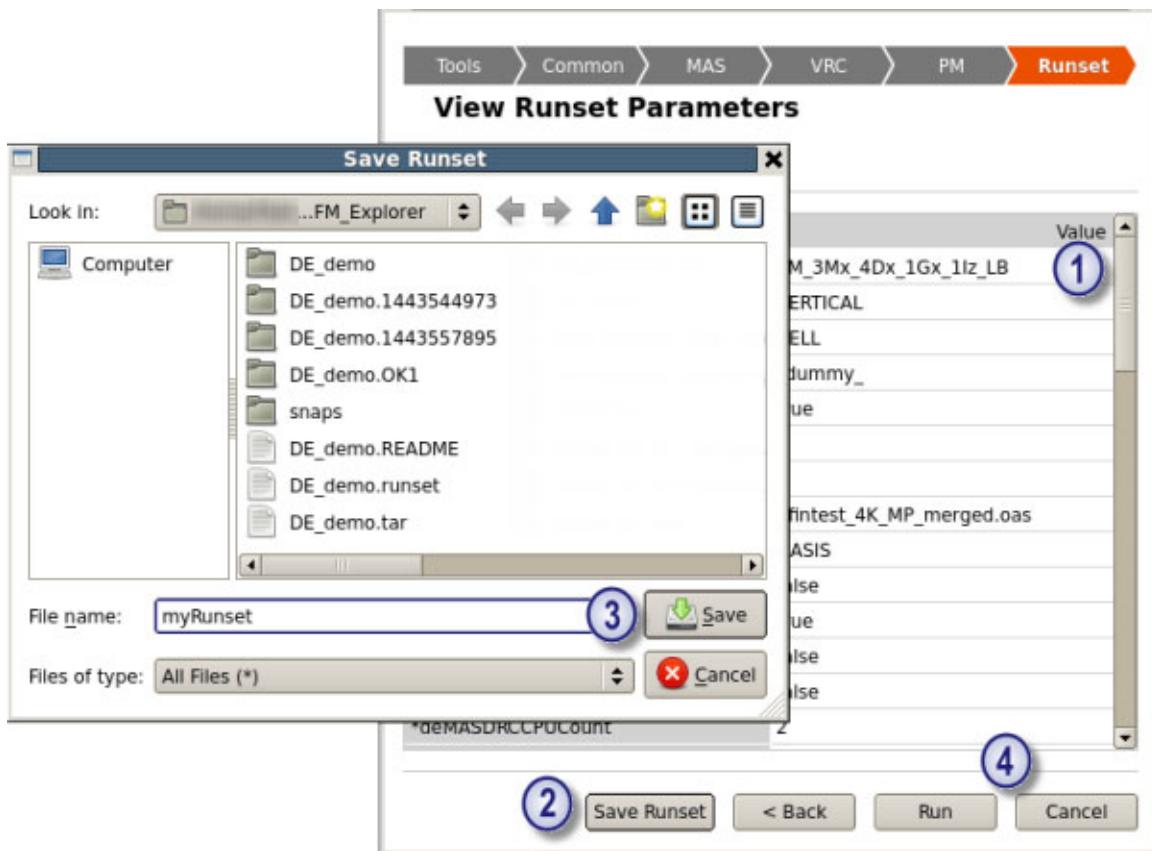


- b. Enter multi-threading and hyperscaling information. Follow the directions given in “[Single-Threaded, Multi-Threaded, and Distributed \(MTflex\) Run Options](#)” on page 140.
 - c. Optional — The following tool tabs require additional setup information.
 - o **MAS Tab** — Enter a filename and location for the **Batch Reporting Configuration File**. This is a template file for HTML reporting.
If the PDK follows the necessary structure, then this filename is automatically filled in. For more information, see “[DFM Explorer PDK Structure](#)” on page 133 and “[DFM HTML Reporting](#)” in the *Calibre RVE User’s Manual*.
 - o **VRC Tab** — Enter a filename in the **Results Oasis** field. This file stores single via error markers.
5. Review the setup information and run DFM Explorer.

- a. Scroll down the tab and review the entered information (1). Incorrect information is highlighted in red.

Tip  If you find an entry that is red, double-click on it with the left mouse button to go to the corresponding field and make the necessary correction.

- b. Click **Save Runset** (2). This opens the Save Runset window. If you choose not to save the runset, it is automatically saved as *de_runset*. This file is saved to the output directory and overwritten every time the tool runs.
- c. Name your runset file and save it (3).
- d. Click **Run** to execute the DFM Explorer run (4).



Results

If you selected **Show Results in Browser** in the tools window, your default browser opens automatically and displays the run results. As DFM Explorer executes each analysis run, the browser is updated.

- The DFM Explorer Reports section (1) displays a summary of input information. This is information you entered in the **Common** tab.
- The Report Index (2) section shows if the job execution was successful and if the analysis passed or failed the design criteria.
- Each Report for *tool_name* (3) section shows more detailed summary of the run status as well as the location for the results database for that tool (LFD, MAS, VRC, and PM).

- If a tool is skipped (4), this is indicated in the report.

DFM Explorer Reports	
Layout System	OASIS
Layout PATH	./fintest_4K_MP_merged.oas
Technology Directory	10nm/PDK_1.0.0.2/S00-V1.0.0.2/DFM
Project Name	demo
Top Cell	*
BEOL Stack	9M_3Mx_4Dx_1Gx_1Iz_LB
Exclude Cell	_dummy_
Run Method	LMP

Report Index		
Tool	Job Status	Result
MAS	SUCCEEDED	PASS
LFD	SKIPPED	
PM	SUCCEEDED	PASS
VRC	SUCCEEDED	FAIL

Report for MAS

Job Status	SUCCEEDED
Result	PASS
Criteria	must be greater than or equal to 0.95 for each rules
Output Dir	'DFM_Explorer/DE_demo/MAS'
PDK	10nm/PDK_1.0.0.2/S00-V1.0.0.2/DFM
Variables	'10nm/PDK_1.0.0.2/S00-V1.0.0.2/DFM'
Validation Script	'10nm/PDK_1.0.0.2/S00-V1.0.0.2/DFM'
CPU Count	2
RDB Dir	/DE_demo/MAS/demo/rdb

Report for LFD

Job Status	SKIPPED
------------	---------

Chapter 7

Fill Layers in Calibre YieldEnhancer

The DFM Spec Fill Optimizer, DFM Spec Fill Shape, DFM Spec Fill, and DFM Fill rule file elements are used in modern fill solutions. These elements comprise the Calibre SmartFill solution.

Calibre SmartFill capabilities accommodate the latest chip manufacturing techniques and considerations such as mask density, double and triple patterning, stress analysis, parasitic effects and timing closure, and so forth. The generated fill is “correct by construction” and, therefore, DRC clean.

For most recent technology nodes, fill rule file development is performed by foundries. Hence, unless you are doing foundry rule file development, you need not concern yourself with creating fill rules if your foundry provides them.

You may need to perform limited changes (ECO) to fill if the design changes. This can be accomplished using the ecofill utility. Technology files and configuration information are available from your Calibre sales support team.

Calibre SmartFill Flow [158](#)

Calibre SmartFill Flow

Adding fill to your layout using Calibre SmartFill begins with a DRC-clean layout and ends with a DRC-clean layout with fill.

Figure 7-1 shows the basic flow.

Figure 7-1. Basic Flow for Adding Fill

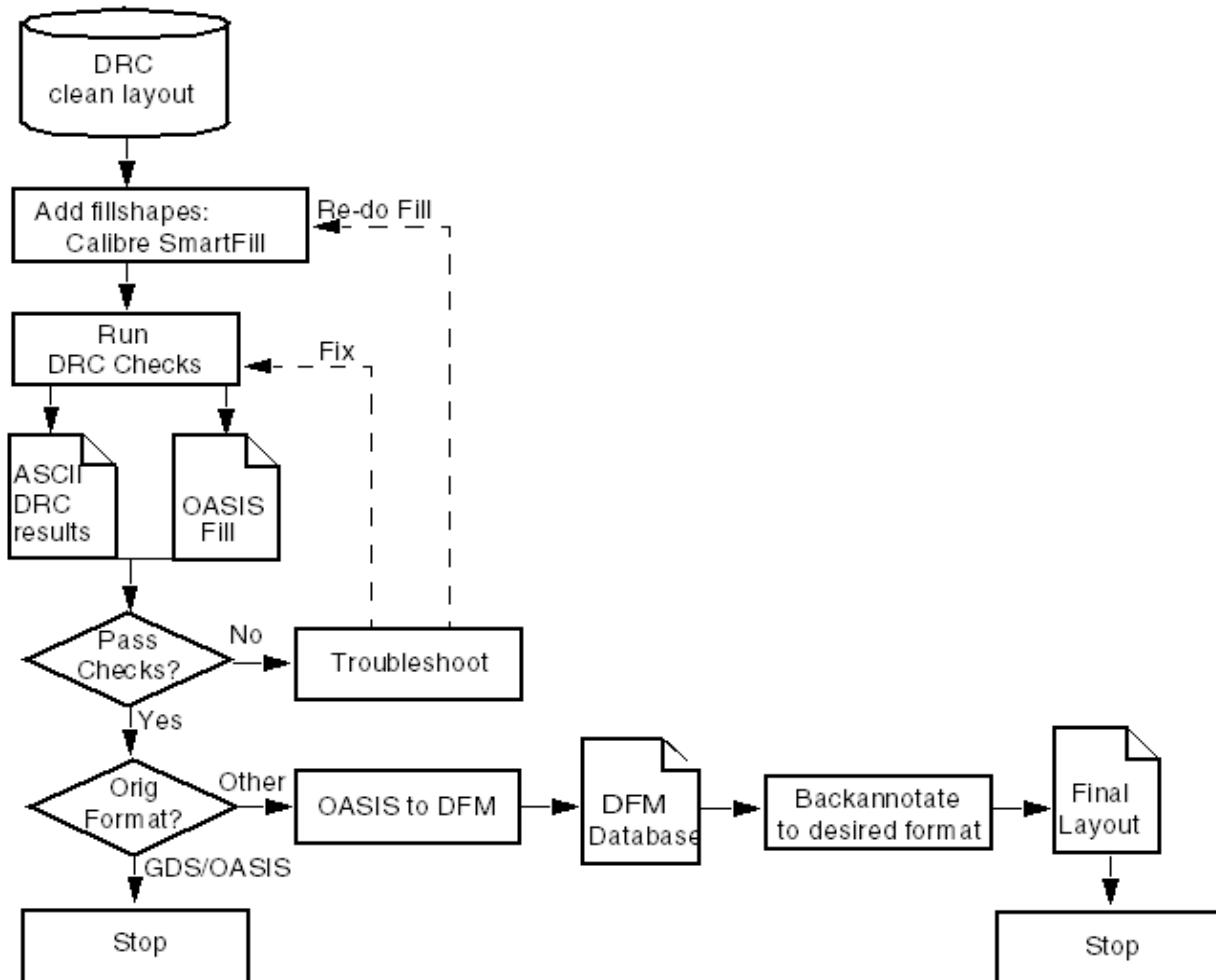


Table 7-1. Stages of Adding Fill

Stage	Description
1	Create or access a rule file for running Calibre SmartFill.
2	After your design is DRC clean, run Calibre SmartFill to add fill. Your run mode choices are: <ul style="list-style-type: none">• Calibre DFM Command Line Invocation• Invocation of Calibre DFM from Calibre Interactive

Table 7-1. Stages of Adding Fill (cont.)

Stage	Description
3	Use Calibre RVE for DFM in conjunction with your layout viewer or editor to review any DRC errors. Depending on the number and type of errors you discover, you either repair the problems in your layout viewer, or modify your fill specification and generate a revised fill layer.
4	After DRC passes on all layers, save the layout and fill layer(s).
5	Backannotate the fill into your original design. See “ Via and Fill Backannotation ” in the <i>Calibre Layout Comparison and Translation Guide</i> .

If engineering change orders (ECOs) necessitate regenerating sections of your fill layers, consider using [ecofill](#) for this purpose.

Tips for Adding Fill

DFM Fill Spec and DFM Fill Spec Shape statements can have a significant impact on both the effectiveness of the fill solution and the run-time performance of the Calibre SmartFill toolset. Certain practices can provide better results.

The secondary keywords mentioned in this discussion are for [DFM Spec Fill](#). The following table shows relative performance of these keywords.

Table 7-2. Performance Based on DFM Spec Fill Shape Type

Fill Shape Type	Performance
STRETCHFILL	BEST
RECTFILL	INTERMEDIATE
POLYFILL	WORST

- Whenever possible, use the STRETCHFILL and AUTORotate keywords, as they have been shown to produce the best results and the shortest run times.
- Fill constraints should conform to foundry requirements.
- Design your fill sequence order from largest fill shape to smallest fill shape. Here, largest and smallest are in terms of area.

Note

 When comparing fill shape sizes, be sure to include the STEP and the SPACE TO LAYER. Thus, shape size is actually SHAPE extent + SHAPE STEP + SPACE TO LAYER. You may need to add a smaller fill shape to a layer first if it has either a large STEP, a large SPACE, or both.

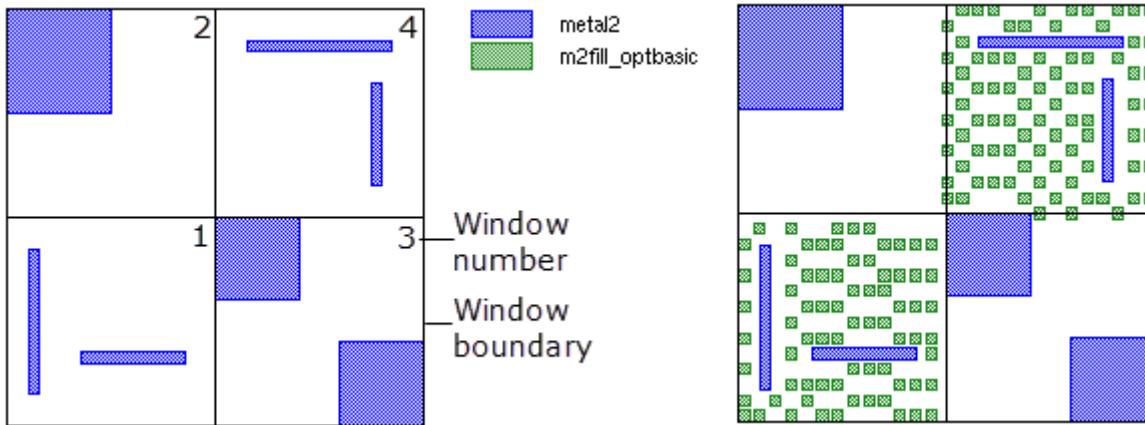
- In general, EFFORT should be specified when inserting the maximum amount of fill in a given area.

- Use INSIDE OF LAYER to specify design extents or module boundaries. Use SPACE for all other placement control needs.
- To obtain the smallest file sizes when the output format is GDS, compress the resulting fill layers using the AUTOREF option for the [DFM RDB GDS](#) statement.

Preventing Empty Fill Areas

In some cases the density of certain regions in a design already meets the density requirement when measured in large window, and as a result, no fill is added to such windows. This can cause areas with no fill, which should be avoided. There are two methods for preventing empty fill areas.

Suppose the combined fill and metal2 density should be 25% or greater. In the following figure, windows 2 and 3 meet this requirement in the original design, which is shown on the left. By default, no fill is added to these windows, as shown on the right. The fill code is shown below the figure. In most designs the large areas with no fill are undesirable despite meeting the density constraint for the window.



```
// Fill with basic optimizer and no handling for empty fill regions
DFM SPEC FILL OPTIMIZER "OptBasic" metal2
    [ ( AREA(metal2) + AREA(_FILL_) ) / AREA() ] >= 0.25

DFM SPEC FILL SHAPE "Sh1"
    RECTFILL 0 0 0.1 0.1 STEP 0.05
    SPACE 0.05 metal2
    OPTIMIZER OptBasic

DFM SPEC FILL OPT_BASIC
    WINDOW 2
    RDB fill_basic.rdb // for debug only, to report fill statistics
    FILLSHAPE OUTPUT m2 "Sh1"

m2fill_optbasic = DFM FILL OPT_BASIC
```

Suppose any empty region of equal to or larger than an 0.5x0.5 user unit square should have at least one fill shape. There are two methods for accomplishing this:

Table 7-3. Methods to Prevent Empty Fill Areas

Method	Comments
Method 1: Use the NOEMPTY keyword in DFM Spec Fill Optimizer	Requires use of the TOMAX keyword and a less than (<) optimization limit. Simple to implement, but the final density may be greater than the density limit in some windows.
Method 2: Define a second fill shape with STEP, SPACE, and SHAPESPACE settings that fill the empty regions as required.	Requires a second fill shape, but can be used with a greater than (>) optimization limit. The second fill shape can be different than the main fill shape.

Method 1: NOEMPTY and TOMAX in DFM Spec Fill Optimizer

The NOEMPTY keyword in DFM Spec Fill Optimizer defines a minimum empty area constraint. Fill shapes are added as needed to satisfy the empty area constraint. When such fill shapes are added to a window that already satisfies the TOMAX optimization limit, the result exceeds the optimization limit.

```
// Method using NOEMPTY and TOMAX in DFM Spec Fill Optimizer
VARIABLE m2_noempty_size 0.5 // Area bigger than 0.5x0.5
                                // should not be empty
DFM SPEC FILL OPTIMIZER "OptNoempty" metal2
[ ( AREA(metal2) + AREA(_FILL_) ) / AREA() ] < 0.25 TOMAX
    NOEMPTY m2_noempty_size

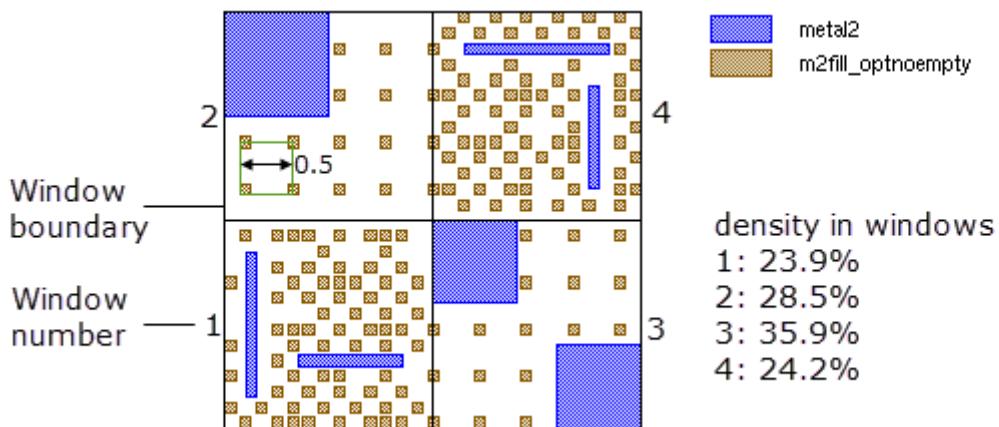
DFM SPEC FILL SHAPE "Sh2"
    RECTFILL 0 0 0.1 0.1 STEP 0.05
    SPACE 0.05 metal2
        OPTIMIZER OptNoempty

DFM SPEC FILL OPT_NOEMPTY
    WINDOW 2
        RDB fill_noempty.rdb // For debug only, to report fill statistics
        FILLSHAPE OUTPUT m2 "Sh2"

m2fill_optnoempty = DFM FILL OPT_NOEMPTY
```

The result is shown in the next figure. In windows 2 and 3, where fill was added to satisfy the NOEMPTY constraint, the final density exceeds the maximum density constraint.

Figure 7-2. Fill Results With NOEMPTY and TOMAX

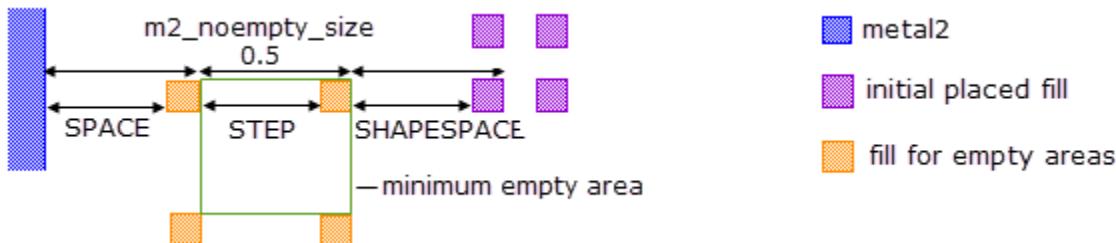


Method 2: Fill Shape for Empty Regions

This method adds fill to empty areas by defining a second fill shape with STEP, SPACE, and SHAPESPACE settings that insure the shape is only placed in empty areas of the given size that remain after initial fill is placed. This is the definition of the second fill shape:

```
VARIABLE m2_noempty_size 0.5 // Area bigger than 0.5x0.5
                                // should not be empty
VARIABLE shape_size 0.1
VARIABLE spacing (m2_noempty_size - shape_size)

DFM SPEC FILL SHAPE "Empty_Regions"
    RECTFILL 0 0 shape_size shape_size
    STEP spacing           // Shape to shape spacing for current shape
    SPACE spacing metal2   // Spacing to metal2 for current shape
    SHAPESPACE spacing spacing // Spacing to previous and later placed fill
    UPDATE "OptBasic"       // UPDATE is not needed for proper fill
                            // placement in this case, but it updates
                            // the optimizer for RDB reporting.
```



This is the complete fill code:

```
VARIABLE m2_noempty_size 0.5 // Area bigger than 0.5x0.5
                                // should not be empty
VARIABLE shape_size 0.1
VARIABLE spacing (m2_noempty_size - shape_size)
```

```

// basic fill shape and optimizer
DFM SPEC FILL OPTIMIZER "OptBasic" metal2
[ ( AREA(metal2) + AREA(_FILL_) ) / AREA() ] >= 0.25

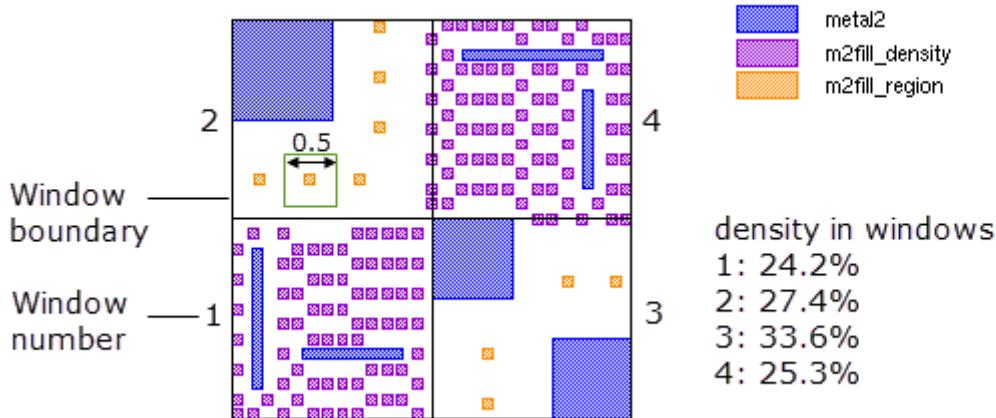
DFM SPEC FILL SHAPE "Sh1"
RECTFILL 0 0 shape_size shape_size
STEP 0.05
SPACE 0.05 metal2
OPTIMIZER OptBasic

// second fill shape for empty regions
DFM SPEC FILL SHAPE "Empty_Regions"
RECTFILL 0 0 shape_size shape_size
STEP spacing           // Shape to shape spacing for current shape
SPACE spacing metal2   // Spacing to metal2 for current shape
SHAPESPACER spacing spacing // Spacing to previous and later placed fill
UPDATE "OptBasic"       // UPDATE is not need for proper fill
                         // placement in this case, but it updates
                         // the optimizer for RDB reporting.

DFM SPEC FILL OPT_BASIC_AND_EMPTY_FILL
WINDOW 2
RDB fill_regions.rdb      // For debug only, to report fill statistics
FILLSHAPE OUTPUT m2_density "Sh1"
FILLSHAPE OUTPUT m2_regions "Empty_Regions"
REPEAT 1 EFFORT 2         // REPEAT and EFFORT improve fill placement
m2fill_density = DFM FILL OPT_BASIC_AND_EMPTY_FILL m2_density
m2fill_region = DFM FILL OPT_BASIC_AND_EMPTY_FILL m2_regions

```

Figure 7-3. Fill Results With Shape for Empty Regions



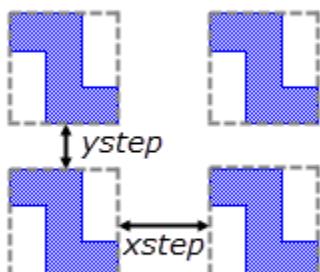
The two fill shapes were output to different layers for clarity in the example, but this is not necessary. To output fill to one layer, use one DFM Fill operation:

```
m2fill_allregion = DFM FILL OPT_BASIC_AND_EMPTY_FILL
```

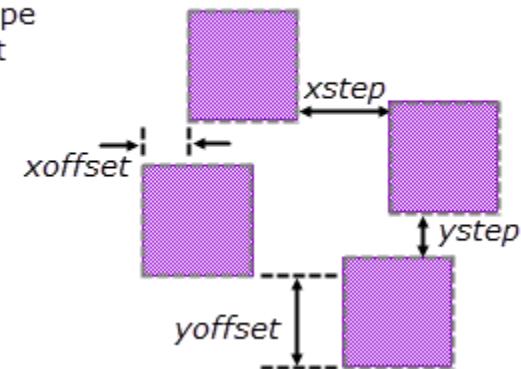
Defining the Fill Pattern: Simple or Staggered

The fill pattern is determined by the STEP and OFFSET keywords in DFM Spec Fill or DFM Spec Fill Shape. A simple rectangular array uses the STEP keyword only. A staggered array is defined with both the STEP and OFFSET keywords.

Rectangular Array



Staggered Array



Rectangular Array	Staggered Array
<pre>DFM SPEC FILL specZ FILLSHAPE OUTPUT Z "zee" STEP 0.1 0.05 // xstep ystep</pre>	<pre>DFM SPEC FILL SHAPE shapeAoffset RECTFILL 0 0 0.12 0.12 STEP 0.1 0.05 // xstep ystep OFFSET 0.05 0.1 // xoffset yoffset</pre>

STEP and OFFSET are not used in STRIPE fill.

Procedure

- (Recommended) Define the separation between adjacent fill shapes with the STEP keyword.
 - STEP *step* — Separation *step* in both the x and y directions.
 - STEP *xstep ystep* — Separation *xstep* and *ystep* in the x and y directions, respectively.

Dimensions are in user units and are measured between the rectangular extents of the fill shapes. The default is 1 dbu, which is not recommended. The STEP keyword is placed in DFM Spec Fill or DFM Spec Fill Shape, after the definition of the fill shape.

```
DFM SPEC FILL specZ
  FILLSHAPE OUTPUT Z "zee"
    STEP 0.1 0.05 // xstep ystep
```

- (Optional) Specify an offset between adjacent fill shapes with the OFFSET keyword. This results in a staggered fill pattern.

- **OFFSET** *offset* — Offset *offset* in both the x and y directions.
- **OFFSET** *xoffset yoffset* — Offsets *xoffset* and *yoffset* in the x and y directions, respectively.

Dimensions are in user units and are measured between the rectangular extents of the fill shapes. The default offset is 0. The **OFFSET** keyword is placed in DFM Spec Fill or DFM Spec Fill Shape, after the definition of the fill shape.

```
DFM SPEC FILL SHAPE shapeAoffset
    RECTFILL 0 0 0.12 0.12
        STEP 0.1 0.05 // xstep ystep
        OFFSET 0.05 0.1 // xoffset yoffset
```

3. (Optional) Specify spacing to fill shapes in *different* fill regions or *different* fill patterns using the keywords in the following table.

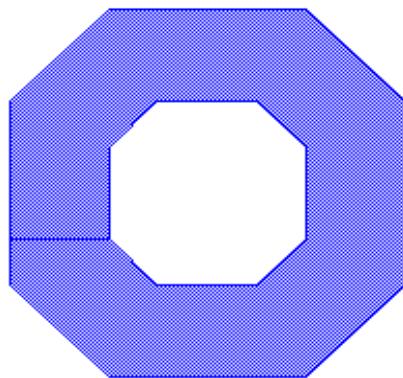
Keyword	Definition
SETBACK	Specifies the minimum spacing between fill shapes of the same fill pattern but in <i>different</i> fill regions. The STEP spacing is used if SETBACK is not specified.
ALLOW	Specifies valid spacing ranges for projecting fill shapes of the same fill pattern but in <i>different</i> fill regions.
SHAPESPACE	Specifies spacing between shapes in the current fill pattern and other fill geometry (fill placed previously or later). The effect of SHAPESPACE is to create a keep-out area for shapes in <i>different</i> fill patterns.

Defining Complex Polygons for Fill Shapes

Fill shapes can have as many vertices as needed. Shapes that are not rectangular are complex.

Complex polygon fill shapes may not self-intersect.

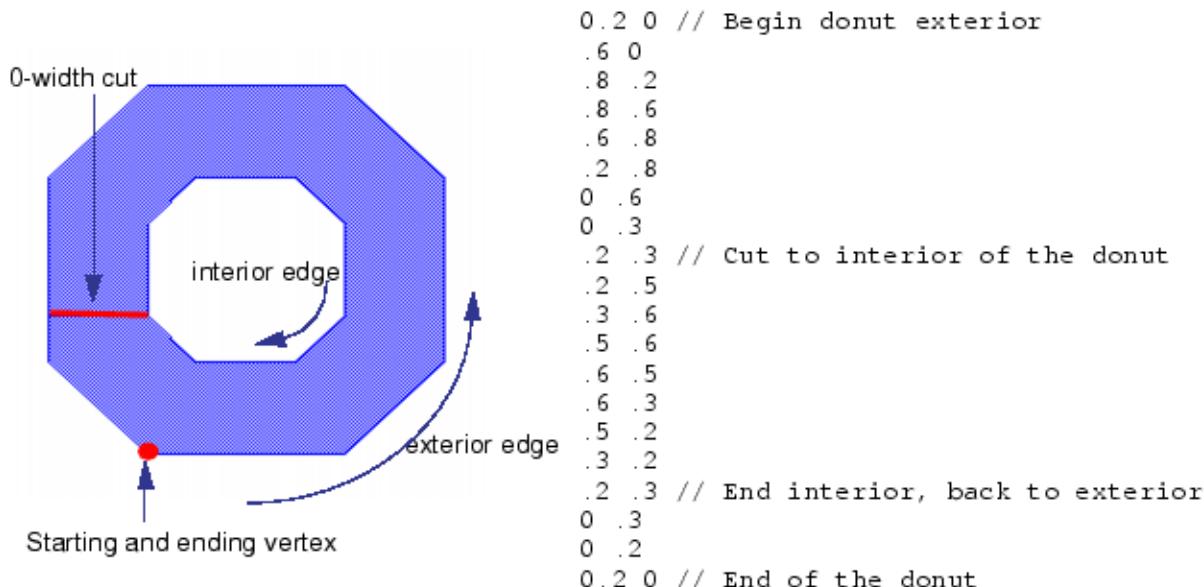
Figure 7-4. Donut Shaped Fill Shape



Procedure

1. Begin the fill shape with the DFM Spec Fill [Shape] POLYFILL keyword.
2. Specify each of the vertices of the polygon.
 - Each coordinate pair defines a single vertex of a closed polygon. The last vertex in the polygon may or may not match the first one. If it does not, the polygon is automatically closed. The last vertex is connected to the first one.
 - Polygons with holes in them can be defined by using 0-width cuts for which the edge-in is the same as edge-out, but in the opposite direction.
 - When creating a polygon with holes in it, vertices of the polygon exterior should be enumerated in a counter-clockwise direction, whereas inside holes should be enumerated in a clockwise direction. This prevents cut lines from intersecting.

Figure 7-5. How to Create the Donut Shaped Fill



3. Specify the DFM Spec Fill [Shape] STEP keyword set as needed.
4. Specify the DFM Spec Fill [Shape] OFFSET keyword set as needed.

Chapter 8

Calibre YieldEnhancer ecofill for Fill with Design ECOs

When changes are made to the design through an engineering change order (ECO) it is preferable to maintain the existing fill design and implement changes only to the affected layout design areas. This is because reapplying fill to the entire layout design involves long run times and possible changes to the entire fill design. Making changes to the fill only in the regions surrounding the ECO area minimizes these impacts.

Most ECO changes also require removing fill shapes. This can involve flattening the fill hierarchy, resulting in an increase in the fill layout database size. Minimizing this side effect is desirable.

The Calibre YieldEnhancer ecofill utility provides a solution for these problems by minimizing fill regeneration time, eliminating the file size increase due to flattening the layout database, minimizing the timing impact to tapeout, and reducing cost impact to masks. It does this by doing the following:

- Flattening the minimum number of fill instances at the lowest required level of the fill hierarchy, removing only those fill shapes that are in conflict with the new ECO design changes.
- Generating fill only in areas where fill shapes were removed, rather than refilling the entire chip.

ECO Fill Flow Using the ecofill Utility	167
ecofill	169

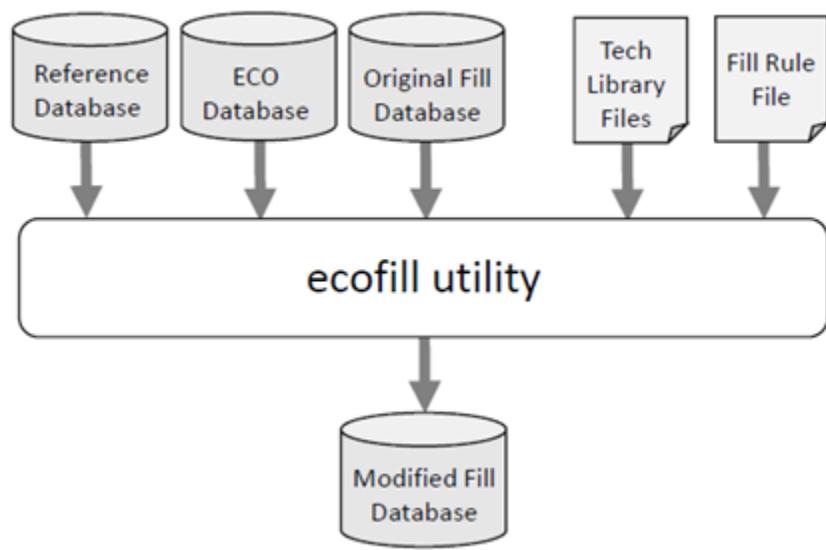
ECO Fill Flow Using the ecofill Utility

The ecofill utility is recommended for adjusting fill adjacent to small ECO changes on metal layers.

The ecofill utility uses a number of Calibre products to produce a final result. You are insulated from interacting with them directly, but they are called in the background and require these licenses: Calibre YieldEnhancer, Calibre nmDRC-H, and Calibre DESIGNrev.

The following figure shows the basic data flow for using the ecofill utility.

Figure 8-1. ECO Flow With the ecofill Utility



The data elements are these:

- **Reference Database** — Original layout database in GDS or OASIS format.
- **ECO Database** — Layout database with the ECO changes, in GDS or OASIS format.
- **Original Fill Database** — Fill database generated for the Reference Database, in GDS or OASIS format.
- **Tech Library Files** — Technology-specific files used by the ecofill utility. The library includes the file *techlib.tcl* containing Tcl procedures, and other necessary files.

The ecofill technology library may be supplied by your foundry. If not, instructions for creating it are available from your Siemens EDA technical sales representative.

- **Fill Rule File** — Rule file containing the commands necessary to generate fill shapes. This file is typically supplied by the foundry.
- **Modified Fill Database** — Revised fill database in GDS or OASIS format. This database is ready for use with the ECO Database.

ecofill

Modifies existing generated fill based on design ECO changes.

Usage

Syntax 1

```
ecofill eco_db ref_db fill_db rule_file [ecofill_options]
```

Syntax 2

```
ecofill {-eco_db eco_db} {-ref_db ref_db} {-fill_db fill_db} {-rule_file rule_file}  
[ecofill_options]
```

Syntax 3

ecofill ...

All or some arguments are specified with environment variables.

Option Syntax

The *ecofill_options* argument has this syntax:

```
[-wrapper scriptname] [-no_refill]  
[-eco_top eco_top] [-ref_top ref_top] [-fill_top fill_top]  
[-log pathname] [-newfill new_fill_db_name] [-rundir rundir_name]  
[-fill_rmv_dir xor_file_path | -fill_rmv filter_shapes_path]  
[-pre_proc] [-post_proc] [-stop_after_xor]  
[-output_format [OASIS | GDS]]  
[-turbo [number_of_processors] [-hyper]]
```

Arguments

- **-eco_db *eco_db***

Required argument set that specifies the changed design (ECO) database without any fill. You can specify a GDS or OASIS database. In Syntax 1, the **-eco_db** keyword may be omitted, but *eco_db* must be the first argument to the command.

- **-ref_db *ref_db***

Required argument set that specifies the original design database without any fill. You can specify a GDS or OASIS database. In Syntax 1, the **-ref_db** keyword may be omitted, but *ref_db* must be the second argument to the command.

- **-fill_db *fill_db***

Required argument set that specifies the original fill database. You can specify a GDS or OASIS database. This database is associated with the *ref_db* design. In Syntax 1, the **-fill_db** keyword may be omitted, but *fill_db* must be the third argument to the command.

- **-rule_file rule_file**

Required argument set that specifies the rule file that was used to produce the original fill database (*fill_db*). In Syntax 1, the **-rule_file** keyword may be omitted, but **rule_file** must be the fourth argument to the command.

- **-wrapper scriptname**

Optional argument set that specifies a script to be run by ecofill in order to produce the modified output fill database. The script can perform various ancillary tasks in support of the ecofill run. The script must contain a calibre -drc invocation with an appropriate rule file to produce the revised fill.

- **-eco_top eco_top**

Optional argument set that specifies the top cell name for the ECO database (*eco_db*). An error is issued if the specified top cell is not found. If not specified, the top cell name is determined automatically.

- **-ref_top ref_top**

Optional argument set that specifies the top cell name for the original design database (*ref_db*). An error is issued if the specified top cell is not found. If not specified, the top cell name is determined automatically.

- **-fill_top fill_top**

Optional argument set that specifies the top cell name for the original fill database (*fill_db*). An error is issued if the specified top cell is not found. If not specified, the top cell name is determined automatically.

- **-no_refill**

Optional argument that specifies to stop after removing fill around the ECO affected area and not continue with generating new fill.

- **-log pathname**

Optional argument set that specifies to save the run transcript to the file *pathname*.

- **-newfill new_fill_db_name**

Optional argument set that specifies the name of the output ECO fill database. By default, the output database name is DM_new.

- **-output_format {OASIS | GDS}**

Optional argument set that specifies the output format for databases that are created by the utility. If not specified, the output format is the same as the original fill database format (*fill_db*).

- **-rundir rundir_name**

Optional argument set that specifies the location for running ecofill. Output files and logs are placed in this location. If specified, output files are placed in the directory *rundir_name.<yyyy_time_stamp>*. If not specified, output files are placed in a directory named *rundir.<yyyy_time_stamp>* in the current working directory.

If specified, *rundir_name* cannot be “rundir”.

In either case, a symbolic link is created as follows:

Case	Symbolic Link
Without -rundir or ECO_RUNDIR, and without -fill_rmv_dir or ECO_FILL_RMV_DIR	rundir -> rundir.<yyyy_time_stamp>
With -rundir or ECO_RUNDIR, and without -fill_rmv_dir or ECO_FILL_RMV_DIR	<i>rundir_name</i> -> <i>rundir_name</i> .<yyyy_time_stamp> If <i>rundir_name</i> is not in the current working directory, for example /path/rundir_base, the link is formed as follows: <i>rundir_base</i> -> /path/rundir_base.<yyyy_time_stamp>
With -fill_rmv_dir or ECO_FILL_RMV_DIR	-rundir or ECO_RUNDIR are ignored. rundir -> xor_file_path

- **-fill_rmv_dir xor_file_path**

Optional argument set that causes the utility to search the specified directory for the files that drive the shape removal process. The files must have the default name(s) of the XOR results: *Mnew.\$sfx*, *Mgone.\$sfx*, *DMrmv.\$sfx*. The \$sfx file extension denotes either a GDS file (.gds) or an OASIS file (.oas). It is not necessary for all files to be present. If not used, the utility searches the default output directory or the -rundir directory, if specified.

If specified, *xor_file_path* cannot be “rundir”.

If -fill_rmv_dir is specified, -rundir and ECO_RUNDIR are ignored and *xor_file_path* is used as the run directory.

- **-fill_rmv filter_shapes_path**

Optional argument set that specifies the database containing shapes defining removal regions. This file is used instead of *Mnew.\$sfx* to drive the shape removal step. The \$sfx file extension denotes either a GDS file (.gds) or an OASIS file (.oas).

- **-pre_proc**

Optional argument that specifies to execute a Tcl procedure immediately after the Calibre DESIGNrev layout peek step is complete (which is executed internally by ecofill), but prior to executing any other steps. The Tcl procedure must be named “ecofill_preproc” and must be defined in the *techlib.tcl* file.

- **-post_proc**

Optional argument that specifies to execute a Tcl procedure after all other steps in the utility are completed. The Tcl procedure must be named “ecofill_postproc” and must be defined in the *techlib.tcl* file.

- **-stop_after_xor**

Optional argument that specifies to stop the ecofill utility immediately after completing the XOR step.

- **-turbo [number_of_processors]**

Optional argument set that triggers multi-threaded parallel processing. The *number_of_processors* parameter is a positive integer that specifies the number of CPUs to use. If *number_of_processors* is not specified, ecofill runs on all available CPUs for which there are licenses.

- **-hyper**

Optional keyword that enables hyperscaling mode. Hyperscaling enables the concurrent, parallel execution of rule file operations in order to maximizes CPU usage efficiency. This option can only be specified with -turbo.

Description

The ecofill utility uses the original layout, ECO layout, fill rule file, original fill database, and a technology library to create a new fill database for the ECO layout. The ecofill utility determines the changed regions of the design, removes fill from the changed regions, and generates new fill for the changed regions. This eliminates the need to regenerate fill for the complete design. Use the ecofill utility for relatively small design ECO changes. For example, an acceptable candidate includes minimal routing changes between gates that address a timing or drive strength issue. A poor candidate for the utility would include a circuit block that was completely rerouted and needs a completely new fill generation.

The ECOFILL_TECH environment variable is required for running the ecofill utility. This environment variable specifies the directory containing the ecofill technology library files. The ecofill technology library may be supplied by your foundry. If not, instructions for creating it are available from your Siemens EDA technical sales representative.

There are three possible syntax forms:

- **Syntax 1** — The abbreviated syntax requires that the four required filenames be specified in the exact order shown in the usage section, but the argument switches for the required filenames are not included. The optional arguments may be specified in any order and must include the argument switches.
- **Syntax 2** — The full syntax includes the argument switches for all arguments, and there is no order dependency for the arguments.
- **Syntax 3** — All or some of the arguments are specified with environment variables. If you specify any of the first four required arguments without the argument switch, you have to specify all of them in order. If you specify any of the first four required arguments with an environment variable, all of the other arguments on the command line need the argument switch. A command line argument takes precedence over an environment variable that sets the same argument.

Table 8-1. Environment Variables for ecofill Utility

Environment Variable	Corresponding Command Line Switch and Argument
ECO_ECO_DB	-eco_db <i>eco_db</i>
ECO_REF_DB	-ref_db <i>ref_db</i>
ECO_FILL_DB	-fill_db <i>fill_db</i>
ECO_RULE_FILE	-rule_file <i>rule_file</i>
ECO_WRAPPER	-wrapper <i>scriptname</i>
ECO_ECO_TOP	-eco_top <i>eco_top</i>
ECO_REF_TOP	-ref_top <i>ref_top</i>
ECO_FILL_TOP	-fill_top <i>fill_top</i>
ECO_NO_REFILL	-no_refill
ECO_LOG	-log <i>log_file_name</i>
ECO_NEWFILL	-newfill <i>new_fill_db_name</i>
ECO_OUTPUT_FORMAT	-output_format {OASIS GDS}
ECO_POST_PROC	-post_proc
ECO_PRE_PROC	-pre_proc
ECO_RUNDIR	-rundir <i>rundir_name</i>
ECO_STOP_AFTER_XOR	-stop_after_xor
ECO_FILL_RMV_DIR	-fill_rmv_dir <i>xor_file_path</i>
ECO_FILL_RMV	-fill_rmv <i>filter_shapes_path</i>

The following apply when using the ecofill utility:

- Fill shapes are only removed from **fill_db**. Any fill shapes present in the design databases (**ref_db** or **eco_db**) at lower hierarchy levels are not removed because they are considered part of the design database.
- Mixed database formats are allowed. For example, you can specify a **fill_db** database in OASIS format while specifying a **ref_db** in GDS format.
- Different top cell names are allowed between the different input databases. For example, the **ref_db** may contain the top cell “top_design,” while the **eco_db** may contain “top_eco”, and the **fill_db** may contain “top_fill”.
- The technology library can be configured to make changes to the environment variables ECO_ECO_DB, ECO_REF_DB, and ECO_FILL_DB at any point during the ECO Fill flow.

Examples

Example 1

This is an example of Syntax 1, where only the required files are provided in the required order:

```
ecofill design_eco.gds original_design.gds metalfill.oas \
metal_fill_rules.svrf
```

Example 2

This is an example of Syntax 2, where each file is specified with an option. Because of the options, there is no order dependency:

```
ecofill -eco_db design_eco.gds -fill_db metalfill.oas \
-ref_db original_design.gds -log runtime.log \
-output_format OASIS -rule_file metal_fill_rules.svrf
```

Example 3

This is an example of Syntax 3, where both environment variables and command line options are used:

```
// environment variable settings
setenv ECO_ECO_DB /usr/project/design/design_eco.oas
setenv ECO_FILL_DB /usr/project/design/metalfill.oas
setenv ECO_REF_DB /usr/project/design/original_design.gds
setenv ECO_RULE_FILE /usr/project/design/metal_fill_rules.svrf
setenv ECO_LOG /usr/project/design/runtime.log

// ecofill utility invocation
ecofill -run_dir "/usr/project/design/v2" -log_file runtime_V2.log
```

In this example, the ECO_LOG variable is overridden by the -log_file option.

Chapter 9

Calibre YieldEnhancer PowerVia

Calibre® YieldEnhancer PowerVia is a flow with a utility (powervia) that is specifically designed to provide you with an automated solution for inserting additional vias to help improve the yield and reliability of your designs.

The powervia utility scans your design by net name(s) defined by you. It locates layer intersections of same nets and adds as many vias as possible in these intersections in via arrays that are DRC- and LVS-clean.

The powervia utility resides inside the Calibre tree and requires licenses for the following products and tools:

- Calibre YieldEnhancer.
- Calibre nmDRC-H.
- Calibre DESIGNrev — Required when either of the following variable conditions exist:
 - If the LAYOUT_PRECISION variable is *not* specified.
 - If the MERGED_OUTPUT variable is specified.

Refer to “[powervia](#)” on page 226 for a table of variables used in the flow along with related license requirements.

Refer to the [Calibre Administrator’s Guide](#) for complete licensing information.

Calibre YieldEnhancer PowerVia Flow	176
Running the powervia Utility	179
Via Constraints	180
Layer Stack Definition	181
Spacing Between Vias	185
Connectivity-Based Spacing for Vx to Vx-1 Layer Vias	187
Spacing from BAR Vias	189
Same-Layer Spacing Rule	190
Connectivity-Based Spacing for Same-Layer Vias	192
Spacing by PRL for Same-Layer Vias	193
Exclusion Areas	195
Via Count Rules Filtering	196
Original Via Layers Extraction for Extended Via Types	198
Calibre YieldEnhancer PowerVia Files and Specifications.....	201
Via Variables File Specifications	201

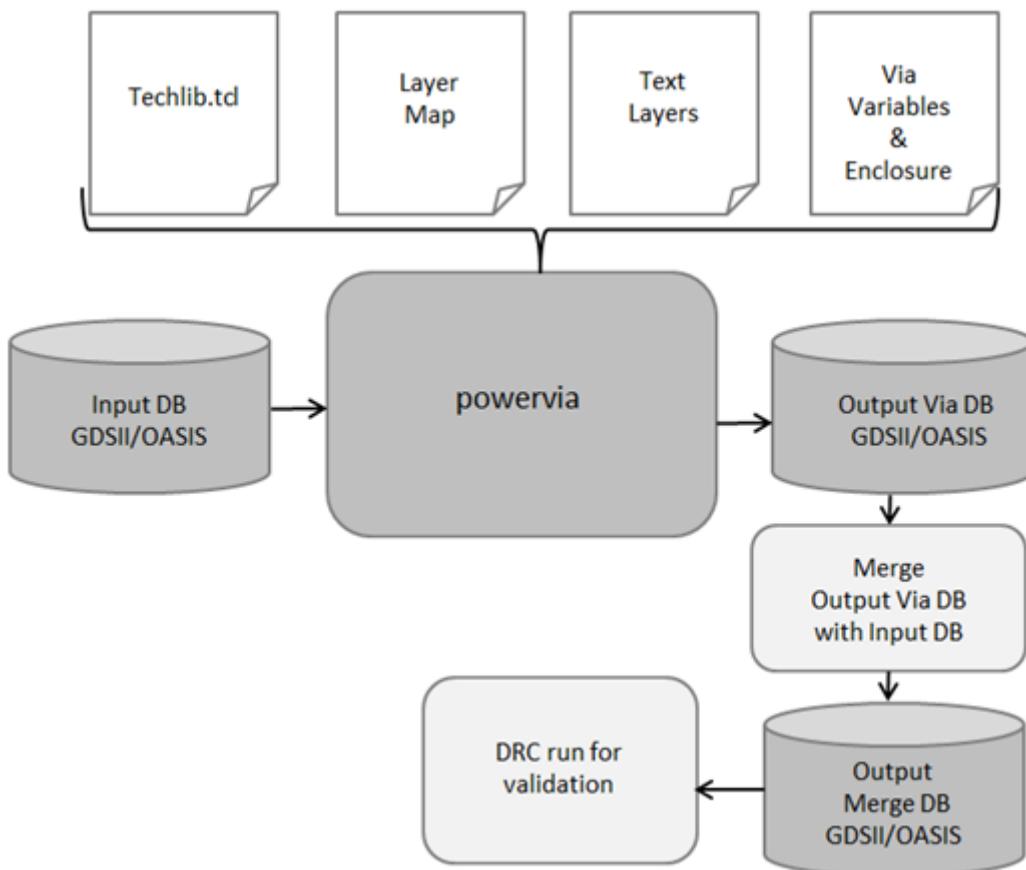
Enclosure Template File Specifications	207
Generated Enclosure Template File Specifications	208
Calibre PERC Flow File Specifications	209
Backannotation Flow Specifications	210
Net Name Specifications	213
Calibre YieldEnhancer PowerVia techlib.tcl File Specifications	214
Calibre YieldEnhancer PowerVia Command Reference	225
powervia	226
powervia Variables	230

Calibre YieldEnhancer PowerVia Flow

This automated flow uses the powervia utility to insert additional vias to reduce IR drop and EM issues in a design.

You can use the Calibre YieldEnhancer PowerVia flow as a generic solution for adding additional vias in place and route designs and custom layout regions. This flow can be run at the block or IP level, and at the chip level for LVS- and DRC-clean designs. Refer to your Siemens EDA representative for Calibre run mode guidelines that are specific to your design.

[Figure 9-1](#) shows the basic data flow for using the powervia utility.

Figure 9-1. Calibre Yield Enhancer PowerVia Flow with the powervia Utility

The inputs are a text annotated GDSII or OASIS database streamed out from your design environment, and a technology and information file (*techlib.tcl* file), layer map, text layer, via variable files, and design and foundry specific information. The powervia utility scans and identifies the nets in the design and the locations in which to insert additional DRC-clean vias.

The output is an updated via database in GDSII or OASIS format that can be merged with the input layout database and then validated in a full Calibre nmDRC-H run. You can analyze the Calibre transcript, view the summary report, and use Calibre DESIGNrev or other layout viewer to display the inserted vias and layers of interest output from the flow.

These are the basic input data elements for the flow:

Note

 Your input files and their content depend on your foundry process and node and may vary from the files listed. Requirements and dependencies are noted. See your Siemens EDA representative for assistance with customizing your inputs for the Calibre YieldEnhancer PowerVia flow.

- **Techlib.tcl** — The technology-specific information and controls are contained in the *techlib.tcl* file (Tcl procedures). A *techlib.tcl* file may be supplied by your foundry. Instructions for creating it are available from your Siemens EDA representative. Required input for all foundry processes.
See “[Calibre YieldEnhancer PowerVia techlib.tcl File Specifications](#)” on page 214 for information on the variables and procedures in this file.
 - **Input Database** — The input layout database GDSII or OASIS format (text annotated and streamed from a place and route or custom design environment layout tool). Required input for all foundry processes.
 - **Layer Map** — The *layermap.svrf* file that contains the drawn layer definitions for your design. This file is required input for all foundry processes.
 - **Text Layers** — The *text_layers* file that contains the layer connectivity definitions. This file is required input dependent on foundry process.
 - **Via Variables** — The via variables file that contains via size, spacing, and via enclosure information. This file is required input and dependent on your foundry process. See your Siemens EDA representative to determine which input file is used for your foundry process. The differences in behavior when using this input file are as follows:
 - **Case 1 (Predefined Via Variables File)** — If this file is used for your foundry process, the via information is predefined per each metal group and used for the run.
 - **Case 2 (Generated Via Variables File)** — If this file is used for your foundry process, a via variables file is generated from the input via variables file during the run based on the Back-End-of-Line (BEOL) layer stack information. The generated filename uses the input filename appended with “_gen” (<filename>_gen).
- See “[Via Variables File Specifications](#)” on page 201 for syntax specifications.
- **Via Enclosure** — The file that contains the via and enclosure information is defined in an editable template provided by your Siemens EDA representative. This file is required input for all foundry processes. The differences in behavior when using this input file are as follows:
 - **Case 1 (Predefined Via Enclosure Template File)** — If this file is used for your foundry process, you predefine the via and metal width-dependent enclosure constraints for each selected via in your BEOL layer stack. See “[Enclosure Template File Specifications](#)” on page 207 for syntax specifications.
 - **Case 2 (Generated Via Enclosure Template File)** — If this file is used for your foundry process, you specify an input file <enc_filename> with the via enclosure values for each via type in your BEOL layer stack. The via enclosure template file contains all the via parameters for each via type and is generated from the input file during the run. The generated filename uses the input filename appended with “_gen” (<enc_filename>_gen. See “[Generated Enclosure Template File Specifications](#)” on page 208 for syntax specifications.

These are the basic output data elements for the flow:

- **Output Via Database** — The output layout database containing the added vias in GDSII or OASIS format.
- **DRC Results Database** — The Calibre nmDRC-H output file from the via insertion run in GDSII or OASIS format.
- **Calibre Log File** — The Calibre nmDRC-H transcript file from the via insertion run.
- **Via Addition Summary** — The Calibre nmDRC-H summary report from the via insertion run.
- **Incremental DEF File** — (Optional flow.) The generated DEF file used to backannotate the added via enhancements into your original design database. See “[Backannotation Flow Specifications](#)” on page 210.

Running the **powervia** Utility

The **powervia** utility is invoked from the command line along with the required environment settings.

Prerequisites

- All license requirements are met as described in the [Calibre Administrator’s Guide](#).
- All input requirements are met for your foundry flow as described in “[Calibre YieldEnhancer PowerVia Flow](#)” on page 176.

Procedure

1. Set the following environment variable in the command line:

```
setenv POWER_VIA_TECH <path>
```

This specifies the directory path (default) to the required input files. Both full and relative (./) paths to the current directory are supported.

2. Invoke the **powervia** utility in the command line:

```
$MGC_HOME/bin/powervia -turbo
```

This runs the Calibre YieldEnhancer PowerVia flow.

Results

When the run completes, it outputs the layout database containing the added vias, the debug database, and the DRC summary report along with other output data.

Examples

See “[powervia](#)” on page 226 for complete command syntax and example specifications for running the **powervia** utility.

Via Constraints

You can use certain specifications in the Calibre YieldEnhancer PowerVia flow to apply constraints, such as setting the lower layers in the via stack, defining the via-to-via spacing checks, and controlling the via counts in arrays.

Note

 Using the Calibre YieldEnhancer PowerVia flow requires at least one technology file that requires special preparation. Configuration instructions are provided by your foundry or Siemens Digital Industries Software. Contact your Siemens EDA representative for details if needed.

Layer Stack Definition	181
Spacing Between Vias	185
Connectivity-Based Spacing for Vx to Vx-1 Layer Vias	187
Spacing from BAR Vias	189
Same-Layer Spacing Rule	190
Connectivity-Based Spacing for Same-Layer Vias	192
Spacing by PRL for Same-Layer Vias	193
Exclusion Areas	195
Via Count Rules Filtering	196
Original Via Layers Extraction for Extended Via Types	198

Layer Stack Definition

The layer definitions for the via stack are controlled through Tcl procedures and via variable specifications based on the foundry process used for the Calibre Yield Enhancer PowerVia flow.

Lower Layer Via Specifications	181
Lower Layer Type (V0 case).....	182
Via Addition	183
Insertion of Different Via Types (LRG, BAR, SQUARE)	183

Lower Layer Via Specifications

The lower starting layers for the Calibre YieldEnhancer PowerVia flow can be set by Tcl procedures in the *techlib.tcl* file and via parameters in the via variables files.

By default, the flow converts an encoded layer stack, specified by the BEOL_STACK variable in the *techlib.tcl* file, to a layer stack with groupings and no lower layer specifications. You can add the following statement to the *techlib.tcl* file as a switch to control whether or not the lower layer specifications are used as the starting layers for the flow:

```
proc names_of_lower_layers
```

The Tcl procedure in the statement returns a string that contains the lower metal and via names. For example:

```
proc names_of_lower_layers {} {  
    return "M0i VIA0i"  
}
```

If the statement is not added to the *techlib.tcl* file, the lower layers are ignored by the powervia utility during the processing flow.

When adding the lower layers to the flow, you must specify a second Tcl procedure to add the lower via and metal layer numbers. This procedure is added by the following statement in the *techlib.tcl* file:

```
proc get_output_layer_numbers
```

Additionally, for the added lower layers, you must set the via parameters in your via variables file as follows:

Note

 Depending on your technology flow, you use a predefined or a generated via variables file. If you have questions on the content of these files, configuration instructions are provided by your foundry or Siemens Digital Industries Software. Contact your Siemens EDA representative for details if needed.

- **Via Parameters** — For all technology flows, you must add variable statements that specify the values for width, spacing, and length to your via variables file. For example:

```
VARIABLE V0_WIDTH 0.02
VARIABLE V0_LENGTH 0.02
VARIABLE V0_STEP 0.036
...
...
```

- **Connectivity** — For technology flows using the generated via variables file, you must also add connectivity information to your *text_layers* file.

Lower Layer Type (V0 case)

When lower layer via specifications are used for foundry processes that use the generated via variables file, the via type information must be explicitly defined for these lower layers.

All vias in foundry processes that use the generated via variables file have an associated via type (*Vtype*) with the exception of the lower layer (V0) via. Similarly, the lower layer metal (M0) specifications do not have a corresponding metal type (*Mtype*).

For example, a “Vx” via and “Mx” metal with a type “x” specification are defined in the input via variables file as follows:

```
// Vx variables
VARIABLE Vx_WIDTH value
VARIABLE Vx_LENGTH value
VARIABLE Vx_LENGTH2 value
...
VARIABLE Mx_Mx_SPACE value
VARIABLE Mx_Vx_ENC value
...
...
```

Whereas the lower layer V0 via “VIA0i” and M0 metal “M0i” without a type specification must be defined in the input variables file as follows:

```
VARIABLE VIA0i_WIDTH value
VARIABLE VIA0i_LENGTH value
VARIABLE VIA0i_LENGTH2 value
...
VARIABLE M0i_VIA0i_ENC value
VARIABLE M0i_M0i_SPACE value
...
```

To specify a spacing constraint between a VIA1i via and a lower layer VIA0i via where the VIA1i via has, for example, a type x specification, you can add the following variable statement to the via variables input file:

```
VARIABLE Vx_VIA0i_LOWER_SPACE value
```

Via Addition

The via addition process can be controlled by a variable and parameters defined in the *techlib.tcl* file.

You specify the SELECT_LAYER variable and parameters to choose the mode of operation that controls how vias are added to your design:

- **Mode 1** — Insert vias on *all* layers defined in the layer stack.

To specify this mode, you must add the following line to the *techlib.tcl* file:

```
set SELECT_LAYER "SINGLE"
```

- **Mode 2** — Insert vias *only* on selected layers in the layer stack.

To specify this mode, you must add the following line to the *techlib.tcl* file:

```
set SELECT_LAYER "via1 via2 ... viaK"
```

Based on the control parameters that you provide, the tool performs the via addition and saves the resulting layers in the output database (*via_punch_fill.oas* or *via_punch_fill.gds*).

Insertion of Different Via Types (LRG, BAR, SQUARE)

The Calibre YieldEnhancer PowerVia flow has the capability to insert different via types such as “LRG”, “BAR”, and “SQUARE” within a single run.

By default, the powervia tool only inserts SQUARE via types for the vias listed in the layer stack. These vias are defined by the BEOL_STACK variable and Tcl procedure “proc get_explicit_stack {}” in the *techlib.tcl* file. When different via types are added to the design,

they share the same fill area and are prioritized starting with the addition of LRG, followed by BAR, and then SQUARE via types.

You can optionally specify a different set of via types per layer by using the VIA_TYPES variable in the *techlib.tcl* file. This variable can be specified with the *techlib.tcl* file SELECT_LAYER variable to define the complete set of via types for the run. Refer to the [powervia](#) command reference for a table of variables used in the flow.

The following three examples show usage models for via type insertion and results:

- Example 1:

```
set SELECT_LAYER "SINGLE"
```

Result 1: All vias types are SQUARE

- Example 2:

```
set SELECT_LAYER "SINGLE"
set VIA_TYPE "V4BAR V4"
```

Result 2: All vias types are SQUARE, except V4: V4BAR and SQUARE

- Example 3:

```
set SELECT_LAYER "V1 V3"
set VIA_TYPE "V3BAR V3LRG"
```

Result 3: V1 is SQUARE (by default), V3: BAR and LRG, only (no SQUARE)



Note

If a via is in the SELECT_LAYER list and has any via type(s) in the VIA_TYPE list, only the via type(s) in the VIA_TYPE list are added.

Required parameters for all requested via types must be specified in the via variables and enclosure template files. The design constraints for different via types can vary between foundry processes, and you may need to consult with your Siemens EDA representative to specify them properly in the *techlib.tcl* file. See “[Calibre YieldEnhancer PowerVia Flow](#)” on page 176 for the list of required input files for the flow.

Here are some *techlib.tcl* file Tcl procedures and via type specification guidelines:

- In this procedure, each via type has a different minimum count per metal width and must be specified separately.

```
proc count_for_wide_metal {via}
```

- In this procedure, the exclude area applies to all via types separately.

```
proc get_exclusion_area {layername}
```

The procedure runs twice for each via layer, first to get the exclusion area common for all via types for that via layer, and then for each via type to get the specific exclusion area for the via type (LRG, BAR, and SQUARE).

- proc get_exclusion_area V1A0 — Runs the procedure to apply a common exclude layer for all V1A0 vias.
- proc get_exclusion_area V1A0BAR — Runs the procedure to apply a specific exclude layer area for only V1A0BAR vias.

See also “[Exclusion Areas](#)” on page 195.

- In this procedure, the space constraint applies to all via types, so there is no need to specify a separate constraint for each via type.

```
proc lower_via_space_constraint_needed { }
```

See “[Calibre YieldEnhancer PowerVia techlib.tcl File Specifications](#)” on page 214 for more information on Tcl procedures used in the Calibre YieldEnhancer PowerVia flow.

Spacing Between Vias

The spacing checks between vias on neighboring (upper and lower) via layers in the Calibre YieldEnhancer PowerVia flow can be controlled by adding Tcl procedures to the *techlib.tcl* file and space constraints to the via variables files.

During via insertion, the powervia utility inserts vias starting from the lower via layers and proceeding to the upper via layers. The space constraints between these neighboring via layers can be checked by adding the following statement to the *techlib.tcl* file:

```
proc lower_via_space_constraint_needed
```

DRC spacing checks are performed for the following via pairs, where *Vx* denotes the current via insertion layer:

- *Vx*, *Vx-1_drawn_vias*
- *Vx*, *Vx-1_inserted_vias*
- *Vx*, *Vx+1_drawn_vias*

The Tcl procedure in the statement returns a list of via layers for which spacing checks are needed. For example:

```
proc lower_via_space_constraint_needed {} {  
    return "V1 V3 V5 V6"  
}
```

If the statement is not added to the *techlib.tcl* file, no spacing checks between neighboring via layers are performed.

Because the drawn vias have different types (BAR, LRG, and others) that must also be checked, a second Tcl procedure must be specified to include all the variations of via types for a particular via layer. This procedure is specified by adding the following statement to the *techlib.tcl* file:

```
proc get_via_variations
```

For example:

```
proc get_via_variations {via} {
    if { $via == "VIA1i" || $via == "VIA2i" } {return "$via"}
    return "$via ${via}BAR"
}
```

The spacing distance for each pair of via types is defined in the via variables files depending on your technology flow.

For technology flows that use the predefined via variables file, the following syntax applies (specify one variable statement per line):

```
VARIABLE <UpperViaType>_<LowerViaType>_SPACE <distance_microns>
```

For example, when adding V2, the space constraints are specified as shown:

```
VARIABLE V2_V1_SPACE 0.2
VARIABLE V2_V1BAR_SPACE 0.2
VARIABLE V2_V1LRG_SPACE 0.2
VARIABLE V3_V2_SPACE 0.3
VARIABLE V3BAR_V2_SPACE 0.3
VARIABLE V3LRG_V2_SPACE 0.3
...
...
```

For technology flows that use the generated via variables file, the following syntax applies (specify one variable statement per line):

```
VARIABLE <UpperLayerType>[via_variation]<LowerLayerType>[via_variation]_LOWER_SPACE <distance_microns>
```

When specifying the LOWER_SPACE constraint, use the following format in the via variables file:

- VARIABLE Vxb_Vx_LOWER_SPACE — Spacing to use when the upper layer is Vxb and the lower layer is Vx.
- VARIABLE Vxb_Vxb_LOWER_SPACE — Spacing to use when the upper layer is Vxb and the lower layer is Vxb.

For example, when adding V2, with via layer types V2 = Vx, V1 = Vx, and V3 = Vy, the space constraints are specified as shown:

```
VARIABLE Vx_Vx_LOWER_SPACE 0.2
VARIABLE Vx_VxBAR_LOWER_SPACE 0.2
VARIABLE Vx_VxLRG_LOWER_SPACE 0.3

VARIABLE Vy_Vx_LOWER_SPACE 0.4
VARIABLE VyBAR_Vx_LOWER_SPACE 0.4
VARIABLE VyLRG_Vx_LOWER_SPACE 0.4
...
```

When specifying Vxb space constraints and Vxb BAR and Vxb LRG via variations, use the following format in the via variables file:

- Vxb_Vxb BAR $_SPACE$ — Specifies the space between square Vxb and bar via Vxb (provided that Vxb BAR is defined in “proc get_via_variation” in the *techlib.tcl* file).
- Vxb_Vxb LRG $_SPACE$ — Specifies the space between Vxb and large-square via Vxb (provided that Vxb LRG is defined in “proc get_via_variation” in the *techlib.tcl* file).

Connectivity-Based Spacing for Vx to Vx-1 Layer Vias

The Calibre YieldEnhancer PowerVia flow uses certain variable specifications to perform connectivity-based spacing from upper-via layers to lower-via layers (Vx to Vx-1).

Connectivity-based spacing for Vx to Vx-1 via layers on different nets is triggered by the existence of corresponding variables in the via variables file only if the Vx via is listed in the following *techlib.tcl* procedure:

```
proc lower_via_space_constraint_needed
```

See “[Calibre YieldEnhancer PowerVia techlib.tcl File Specifications](#)” on page 214 for more Tcl procedure examples.

The connectivity-based spacing distance for each pair of via types is defined in the via variables file with the following naming convention:

```
VARIABLE UpperViaType_LowerViaType_DIFFERENT_NET_SPACE distance_microns
```

See “[Via Variables File Specifications](#)” on page 201 for more variable statement examples.

For example, when adding V2, the variables are as follows:

```
VARIABLE V2_V1_DIFFERENT_NET_SPACE 0.2
VARIABLE V2_V1BAR_DIFFERENT_NET_SPACE 0.2
VARIABLE V2_V1LRG_DIFFERENT_NET_SPACE 0.2
...
```

For foundry process flows that generate a via variables file, different naming conventions apply for defining the connectivity-based spacing distance:

```
VARIABLE UpperLayerType[via_variation]  
[via_variation]_LOWER_DIFFERENT_NET_SPACE distance_microns
```

For example, when adding V2, and via types are V2 = Vx, V1 = Vx, and V3 = Vy, the variables are as follows:

```
VARIABLE Vx_Vx_LOWER_DIFFERENT_NET_SPACE 0.2  
VARIABLE Vx_VxBAR_LOWER_DIFFERENT_NET_SPACE 0.2  
VARIABLE Vx_VxLRG_LOWER_DIFFERENT_NET_SPACE 0.3  
  
VARIABLE Vy_Vx_LOWER_DIFFERENT_NET_SPACE 0.4  
VARIABLE VyBAR_Vx_LOWER_DIFFERENT_NET_SPACE 0.4  
VARIABLE VyLRG_Vx_LOWER_DIFFERENT_NET_SPACE 0.4
```

For BAR vias, short- and long-edge spacing distance is enabled if both of the following variables are defined:

```
via1_via2_LOWER_DIFFERENT_NET_SHORTEDGE_SPACE  
via1_via2_LOWER_DIFFERENT_NET_LONGEDGE_SPACE
```

In this case, the following variable is ignored:

```
via1_via2_LOWER_DIFFERENT_NET_SPACE
```

The allowed keyword combinations (for generated via variables flows only) are listed in the following table. The fields (field1, field2) are shown by the colors magenta (m) and green (g), respectively. The format is as follows:

```
via1_via2_field1(m)_field2(g)_SPACE  
via1_via2_LOWER_field1(m)_field2(g)_SPACE
```

Table 9-1. Lower-Via Spacing Keywords

<i>via1/via2</i>	<i>lower via</i>	<i>lower_viaBAR</i>	<i>lower_viaLRG</i>
<i>upper_via</i>	<i>Empty(m) / DIFFERENT_NET(m)</i>	<i>Empty(m) / DIFFERENT_NET(m)</i>	<i>Empty(m) / DIFFERENT_NET(m)</i>
	<i>Empty(g)</i>	<i>SHORTEDGE(g) / LONGEDGE(g) / Empty(g)</i>	<i>Empty(g)</i>
<i>upper_viaBAR</i>	N/A	<i>Empty(m) / DIFFERENT_NET(m)</i>	<i>Empty(m) / DIFFERENT_NET(m)</i>
		<i>SHORTEDGE(g) / LONGEDGE(g) / Empty(g)</i>	<i>SHORTEDGE(g) / LONGEDGE(g) / Empty(g)</i>

Table 9-1. Lower-Via Spacing Keywords (cont.)

<i>via1/via2</i>	<i>lower via</i>	<i>lower_viaBAR</i>	<i>lower_viaLRG</i>
<i>upper_viaLRG</i>	N/A	N/A	Empty (m) / DIFFERENT_NET (m) Empty (g)

Spacing from BAR Vias

The spacing checks between bar (BAR) vias, whether existing in the design or inserted by the Calibre YieldEnhancer PowerVia flow, can be controlled by adding additional space constraints to the via variables files.

Optional variables for spacing parameters enable different short-edge and long-edge spacing from original (existing) bar vias or to inserted bar vias, during via insertion. For bar vias, if the short- and long-edge space variables are set, then the spacing from the long edge and the short edge is maintained by the specified distance (in microns).

The spacing checks from bar vias apply to either of the following conditions:

- *via* (square, BAR, or LRG) is inserted, and *viaBAR* is an original bar via in the design.
- *viaBAR* is inserted, and *via* (square, BAR, or LRG) is an original via in the design.

The following variable statements are used to specify the spacing checks from bar vias:

```
VARIABLE via_viaBAR_SHORTEDGE_SPACE distance_microns
VARIABLE via_viaBAR_LONGEDGE_SPACE distance_microns
```

For example:

```
VARIABLE VIAx_VIAxBAR_SHORTEDGE_SPACE 0.14
VARIABLE VIAx_VIAxBAR_LONGEDGE_SPACE 0.10
```

The short-edge spacing applies for these conditions:

- The variables for spacing checks from bar vias are specified.
- The flow inserts bar vias *or* there is an original bar via.
- Either of the following insertion scenarios exist:
 - The inserted bar via is orthogonal to the original bar via.
 - The short edge of the inserted bar via is facing the short edge of the original bar via.

The tool considers short-edge and long-edge spacing from original or inserted bar vias to different via types: BAR, LRG, and square (lower vias as well). For example, the following

variables specify the spacing from original or inserted bar vias (in magenta) to inserted BAR and LRG vias:

```
VARIABLE VIAxBAR_VIAxBAR_SHORTEDGE_SPACE 0.6
VARIABLE VIAxBAR_VIAxBAR_LONGEDGE_SPACE 0.3
VARIABLE VIAxBAR_VIAxLRG_SHORTEDGE_SPACE 0.4
VARIABLE VIAxBAR_VIAxLRG_LONGEDGE_SPACE 0.3
...
...
```

Note

 The naming convention of the variable statement follows the insertion priority of the different via types, which are prioritized starting with the addition of LRG, followed by BAR, and then SQUARE via types. In the statement, the lower priority via type is specified to the left of the higher priority via type.

The variables for short-edge and long-edge spacing from original bar vias are specified in the via variables file for your process flow. If these variables are not specified, the SPACE variable must be used. See “[Via Variables File Specifications](#)” on page 201.

Same-Layer Spacing Rule

The Calibre YieldEnhancer PowerVia flow has functionality that controls spacing between vias (original and fill vias) located on the same layers. This functionality is available for all supported technology flows.

The same-layer spacing rule follows the priority of via insertion that begins with large, bar, and square via types, respectively. Spacing is kept between the current fill vias and the existing fill vias. The original drawn vias in the design (before insertion) also have different types and are checked by the same-layer spacing rule. To get all the variations for an original via layer, the “get_via_variations” Tcl procedure must be specified in the *techlib.tcl* file.

For example:

```
proc get_via_variations {via} {
    if { $via == "VIA1i" || $via == "VIA2i" } {return "$via"}
    return "$via ${via}BAR"
}
```

See “[Calibre YieldEnhancer PowerVia techlib.tcl File Specifications](#)” on page 214 for more Tcl procedure examples.

The spacing distance for each pair of via types must be defined in the via variables file. Spacing constraints between bar, large, and square vias follow this naming convention:

```
VARIABLE via_viaBAR_SPACE distance_microns
VARIABLE via_viaLRG_SPACE distance_microns
VARIABLE viaBAR_viaLRG_SPACE distance_microns
```

See “[Via Variables File Specifications](#)” on page 201 for more variable statement examples.

For example, when adding V2, the space constraints are specified as follows:

```
VARIABLE V2_V2_SPACE 0.2
VARIABLE V2_V2BAR_SPACE 0.2
VARIABLE V2_V2LRG_SPACE 0.3
VARIABLE V2BAR_V2LRG_SPACE 0.3
```

If the optional variables for short-edge and long-edge spacing are specified as follows (either via1 or via2 is BAR):

```
via1_via2_SHORTEDGE_SPACE
via1_via2_LONGEDGE_SPACE
```

Then the spacing between the original or inserted vias is kept using these variables according to the short- and long-edge directions of the BAR vias, and the following spacing variable is ignored:

```
via1_via2_SPACE
```

For example, when adding V2BAR, optional variables with different distances from short- and long-edge spacing are specified as follows:

```
VARIABLE V2_V2BAR_SHORTEDGE_SPACE 0.2
VARIABLE V2_V2BAR_LONGEDGE_SPACE 0.3

VARIABLE V2BAR_V2LRG_SHORTEDGE_SPACE 0.4
VARIABLE V2BAR_V2LRG_LONGEDGE_SPACE 0.7
```

The supported keywords combinations are listed in the following table. The fields (field1, field2) are shown by the colors magenta (m) and green (g), respectively:

```
via1_via2_field1(m)_field2(g)_SPACE
```

Not applicable (N/A) keyword combinations and “Empty” (no keyword specified) fields are noted.

Table 9-2. Same-Layer Spacing Keywords

via1/via2	via	viaBAR	viaLRG
via	Empty (m) / DIFFERENT_NET (m) Empty (g)	Empty (m) / DIFFERENT_NET (m) SHORTEDGE (g) / LONGEDGE (g) / Empty (g)	Empty (m) / DIFFERENT_NET (m) Empty (g)

Table 9-2. Same-Layer Spacing Keywords (cont.)

<i>via1/via2</i>	<i>via</i>	<i>viaBAR</i>	<i>viaLRG</i>
<i>viaBAR</i>	N/A	<code>Empty (m) / DIFFERENT_NET (m)</code> <code>SHORTEdge (g) / LONGEDGE (g) / Empty (g)</code>	<code>Empty (m) / DIFFERENT_NET (m)</code> <code>SHORTEdge (g) / LONGEDGE (g) / Empty (g)</code>
<i>viaLRG</i>	N/A	N/A	<code>Empty (m) / DIFFERENT_NET (m)</code> <code>Empty (g)</code>

Connectivity-Based Spacing for Same-Layer Vias

The Calibre YieldEnhancer PowerVia flow uses certain variable specifications to enable connectivity-based spacing for same-layer vias on the same nets and on different nets.

When specified, the following optional variable statements enable connectivity-based spacing for same-layer vias:

`via1_via2_DIFFERENT_NET_SPACE` (for all via types, bar, large, and square)

Or both of the following variables:

`via1_via2_DIFFERENT_NET_SHORTEdge_SPACE` (where either via1 or via2 is BAR)

`via1_via2_DIFFERENT_NET_LONGEDGE_SPACE` (where either via1 or via2 is BAR)

When connectivity-based spacing is enabled, the spacing variables for same-layer vias on the same nets and on different nets are specified as follows:

- Via Spacing (Same Nets) — The common `via1_via2_SPACE` variable defines the spacing for vias on the same nets.
- Via Spacing (Different Nets) — The variable `via1_via2_DIFFERENT_NET_SPACE` defines the spacing for vias on different nets (likewise, for the SHORTEdge and LONGEDGE variable specifications).

For BAR vias, if both the SHORTEdge and LONGEDGE variables are specified as follows:

`via1_via2_DIFFERENT_NET_SHORTEdge_SPACE`

`via1_via2_DIFFERENT_NET_LONGEDGE_SPACE`

Then the following variable is ignored:

`via1_via2_DIFFERENT_NET_SPACE`

For example:

For spacing between SQR V2 vias, the optional variables for different net spacing are specified as follows:

```
VARIABLE V2_V2_DIFFERENT_NET_SPACE 0.2 //for different net spacing
VARIABLE V2_V2_SPACE 0.3           //for same net spacing
```

For spacing between SQR and BAR V2 vias, the variables are specified as follows:

```
VARIABLE V2_V2BAR_DIFFERENT_NET_SHORTEDGE_SPACE 0.2 //for different net
spacing from short edge
VARIABLE V2_V2BAR_DIFFERENT_NET_LONGEDGE_SPACE 0.2 //for different net
spacing from long edge

VARIABLE V2_V2BAR_SHORTEDGE_SPACE 0.3 //for same net spacing from short
edge
VARIABLE V2_V2BAR_LONGEDGE_SPACE 0.3 //for same net spacing from long edge
```

For spacing between BAR and LRG V2 vias the variables are specified as follows:

```
VARIABLE V2BAR_V2LRG_DIFFERENT_NET_SHORTEDGE_SPACE 0.5
VARIABLE V2BAR_V2LRG_SHORTEDGE_SPACE 0.6 //for same net spacing from short
edge
VARIABLE V2BAR_V2LRG_DIFFERENT_NET_LONGEDGE_SPACE 0.5 //for different net
spacing from long edge
VARIABLE V2BAR_V2LRG_LONGEDGE_SPACE 0.6 //for same net spacing from long
edge
```

Refer to “[Via Variables File Specifications](#)” on page 201 for more information on via variable specifications.

Spacing by PRL for Same-Layer Vias

Spacing by parallel run length (PRL) is used in the Calibre YieldEnhancer PowerVia flow to control spacing for same-layer vias on the same nets and different nets. The PRL spacing is based on the projection (edge-length extension) of one via on another.

PRL is a measure of how two vias project their neighboring edges upon each other. For same-layer vias, optional variables control PRL-based spacing for all via types (square, large, and bar) on the same nets and different nets.

- PRL Via Spacing (Same Nets):

```
VARIABLE via1_via2_SPACE_PRL "space_value prl_value"
```

- PRL Via Spacing (Different Nets):

```
VARIABLE via1_via2_DIFFERENT_NET_SPACE_PRL "space_value prl_value"
```

When spacing by PRL is used, the behavior is such that the via spacing must be greater than space_value if the PRL between the neighbor vias is greater than prl_value (absolute value).

The following constraints apply to the variable specifications:

- The space_value is a positive number enclosed in quotation marks “ ” with the prl_value. Units are in microns.
- The prl_value must be a *negative* number enclosed in quotation marks “ ” with the space_value. Units are in microns.
- The PRL-based spacing variable for different nets must be specified along with the SPACE variable and its defined value (see details for SPACE and other variables in “[Via Variables File Specifications](#)” on page 201).

The spacing variables for same-layer vias on the same nets and different nets are independent and can be used separately.

- PRL Via Spacing (Same Nets) Only — When PRL-based spacing is specified for vias on the same nets, and the connectivity-based spacing variable (*via1_via2_DIFFERENT_NET_SPACE*) is specified, the PRL-based spacing is enabled for those vias. If the connectivity-based spacing variable is *not* specified, the PRL-based spacing is kept between the vias regardless of their nets.
- PRL Via Spacing (Different Nets) Only — When the PRL-based spacing is specified for vias on different nets, and the connectivity-based spacing variable (*via1_via2_DIFFERENT_NET_SPACE*) is specified, the PRL-based spacing is enabled. If PRL-based spacing is specified for vias on different nets, and the connectivity-based spacing variable is *not* specified, an error message results and the flow exits.

For BAR vias, both the short- and long-edge variables must be defined to enable the PRL-based spacing for same-layer vias on the same nets and different nets.

- PRL BAR Via Spacing (Same Nets):

VARIABLE *via1_via2_SHORTEDGE_SPACE_PRL* “*space_value prl_value*”

VARIABLE *via1_via2_LONGEDGE_SPACE_PRL* “*space_value prl_value*”

- PRL BAR Via Spacing (Different Nets):

VARIABLE

via1_via2_DIFFERENT_NET_SHORTEDGE_SPACE_PRL “*space_value prl_value*”

VARIABLE *via1_via2_DIFFERENT_NET_LONGEDGE_SPACE_PRL*

“*space_value prl_value*”

If the variables for PRL space for different nets and same nets are specified for a via layer that does *not* have connectivity-based spacing, then the flow behaves as follows:

- Same Nets — PRL-based spacing takes precedence regardless of the net specifications.
- Different Nets — The flow exits with a message that the PRL space is defined without the associated space constraint.

For example, if the following variable statements are defined,

```
VARIABLE via_viaBAR_SPACE distance_microns
VARIABLE via_viaBAR_DIFFERENT_NET_SPACE_PRL "space_value prl_value"
```

and the following space variable is *not* defined,

```
via_viaBAR_DIFFERENT_NET_SPACE
```

then the flow exits with an error message:

```
via1_via2_DIFFERENT_NET_SPACE_PRL is defined without associated via1_
via2_DIFFERENT_NET_SPACE.
```

Exclusion Areas

The definitions of exclusion areas for input layers are used by all foundry processes and are controlled in the Calibre YieldEnhancer PowerVia flow by adding a mandatory Tcl procedure to the *techlib.tcl* file.

An exclusion area is a region where the insertion of additional vias is forbidden. Because each via must be fully covered by upper and lower metal, the exclusion area for a particular Vx layer has three components (not all components are required):

- Vx_exclude — Forbidden space for via layer metal
- Mx+1_exclude — Forbidden space for the upper metal layer
- Mx-1_exclude — Forbidden space for the lower layer metal

The Tcl procedure provides an interface between the Calibre YieldEnhancer PowerVia flow and defined exclusion areas for an input layer that is a via (cut) layer or an interconnect layer from the BEOL stack.

```
proc get_exclusion_area {layerName} {}
```

The Tcl procedure returns the definition of an exclusion area for the input layer specified by *layerName*. Only *known* exclusion areas are handled; if an input layer does not have an exclusion area defined, an empty string is returned from the procedure.

Additionally, the returned string after the *layerName* substitution must be a valid argument for the SVRF [Copy](#) layer operation. It can be either an original layer (Example 1) or an expression of original layers enclosed in a set of parenthesis (Example 2) as shown:

- **Example 1** — Where DM1EXCL, DM2EXCL, ... DM9EXCL are original layers:

```
proc get_exclusion_area {layername} {
    set i [regexp -inline {[0-9]+} $layername]
    return "DM${i}EXCL"
}
```

- **Example 2** — Where (DM1EXL OR VNCAP), etc. are valid expressions of original layers:

```
proc get_exclusion_area {layername} {
    set i [regexp -inline {[0-9]+} $layername]
    return "(DM${i}EXCL OR VNCAP)"
}
```

Note

 The same return value must not be defined for the via layer and the enclosure layer, or the tool exits with an error message stating that identical layers are defined for the same operation.

Via Count Rules Filtering

In certain technology flows, DRC rules are used to create filters for the vias that are inserted into the design. You can use the post-filtering ability of the Calibre YieldEnhancer PowerVia flow to apply rule constraint options to check the minimum via count in a via array.

There are two post-filtering options for checking the minimum via count in arrays where vias have been inserted by the powervia utility. The behavior of these options is such that if the via count fails for an array, then only the inserted vias in that array are removed, and the original vias remain. The via-count filter must be the last post-filtering step among all other filters in order to account for all vias present in the design.

The two options for setting the via counts for post-filtering are as follows:

- **Option 1: Set Via Counts in Via Variables** — Specify the minimum via count in the via variables files by setting a single value that serves as a global count constraint regardless of the widths of the metal enclosure layers for the via.

For technology flows that use the predefined via variables file, the following syntax applies:

```
VARIABLE Via_COUNT value
```

Via is a via in the predefined via variables file, and *value* is a positive integer that is the minimum number of vias required within a via array for that via.

For example:

```
VARIABLE VIA1_COUNT 2
...
VARIABLE VIA8_COUNT 4
...
```

For technology flows that use the generated via variables file, the following syntax applies:

```
VARIABLE Vtype_COUNT value
```

Vtype is the foundry notation for the via type corresponding to the via layer number, and *value* is a positive integer that is the minimum number of vias required within a via array for a via of that type.

For example:

```
VARIABLE Vy_COUNT 2
```

- **Option 2: Set Via Counts in Techlib.tcl** — Specify the minimum via count in the *techlib.tcl* file by providing a Tcl proc (proc *count_for_wide_metal*) that returns a width-dependent count for the upper and lower enclosure layers for the via. Each via type has a different minimum count per metal width and must be specified separately. For example:

```
proc count_for_wide_metal {via} {
    switch -exact $via {
        V1 {return [list \
                    "{10 20 2}" \
                    "{20 30 4}" \
                    "{30 6}" \
                    "{10 20 2}" \
                    "{20 30 4}" \
                    "{30 7}" \
                ]}
        }
        V2 {return [list \
                    "{10 20 2}" \
                    "{20 30 4}" \
                    "{30 8}" \
                    "{10 20 2}" \
                    "{20 30 4}" \
                    "{30 9}" \
                ]}
    }
}
```

The Tcl proc returns a list of two elements each enclosed by quotation marks (“ ”). The first of these elements refers to the lower enclosure layer (for example, M1 for V1), and the second refers to the upper enclosure layer (for example, M2 for V1).

For each of the two elements, there is a list of tuples. For example, for V1 and M1, the list of tuples is as follows:

```
{10 20 2} {20 30 4} {30 6}
```

The first two values in a tuple define the width range for the enclosure layer (in microns), and the last value is the via count. The width range includes the beginning value but not the ending value. If only one value is specified for the width range, the width of the enclosure layer must be greater than or equal to this value. For example:

```
{30 6}
```

In this tuple, the width of the enclosure layer must be $>= 30$ microns, and “6” is the minimum count for the via in an array.

Note

-  If both via count options are specified, the first option takes precedence over the second. If neither option is specified, the minimum via count defaults to “1”, and no via-count filtering is applied. The options differ with respect to consideration of width of the enclosure layers for the via.
-

Original Via Layers Extraction for Extended Via Types

For technology flows that use the generated via variables file, the Calibre YieldEnhancer PowerVia flow uses certain specifications and Tcl procedures to extract the original layers for extended via types (BAR1, BAR2,... LRG1, LRG2...).

Note

-  Depending on your technology flow, you use a predefined or a generated via variables file. If you have questions on the content of these files, configuration instructions are provided by your foundry or Siemens Digital Industries Software. Contact your Siemens EDA representative for details if needed.
-

For the generated via variables flow, each original via layer in the technology contains all via types: bar, large, and square. To perform spacing, count checks, and filtering, each via type must be in a separate layer. These layers are derived from the width and length variables in the via variables file. See “[Via Variables File Specifications](#)” on page 201.

The following rules apply when extracting original via layers:

- If more than one BAR or LRG via type is defined (as for extended via types), a separate definition must be created for each via type with a specific (width and length) rectangle layer size.
- If you define only one BAR or LRG via type, the flow behaves as normal without the need for additional definitions.
- All remaining BAR and SQR vias in the original design, other than those with defined dimensions (width and length), default to via type BAR and SQR layers.

The extraction of original layers for extended via types is based on the following *techlib.tcl* Tcl procedure, which includes all variations of via types for a particular via layer.

```
proc get_via_variations{via} {}
```

The following example shows a generic representation for extracting original via layers for extended via types using the “*proc get_via_variations*” Tcl procedure. See “[Calibre](#)

["YieldEnhancer PowerVia techlib.tcl File Specifications"](#) on page 214 for additional Tcl procedures.

```
proc get_via_variations{via} {
    if {$via == VIA1i} {
        return "${via}BAR ${via}BAR2 ...${via}BARn ${via}LRG ${via}LRG2 ... \
                ${via}LRGm"
    }
}
```

This example shows the “proc get_via_variations” Tcl procedure with extended via types for two BAR vias and two LRG vias.

```
proc get_VIA1i_variations{via} {
    if {$via == VIA1i} {
        return "${via}BAR ${via}BAR2 ${via}LRG ${via}LRG2"
    }
}
```

This example shows the “get_via_variations” Tcl procedure with one BAR via type and two LRG vias with an extended via type.

```
proc get_via_variations{via} {
    if {$via == VIA1i} {
        return "${via} ${via}BAR ${via}LRG ${via}LRG2"
    }
}
```

In the generated via variables flow, the inserted vias list can be larger than the original vias list. For this case, you also need to create the via layers that are listed in the inserted vias list but are not listed in the via variations. This behavior exists because the tool creates the derived layers that contain the original and filled vias used in the count checks. This example uses the “proc extended_type_count” Tcl procedure to specify the count of the extended via types.

```
set SELECT_LAYER "Via4i VIA5i"
proc extended_type_count {via_name} {
    if {$via_name == "VIA4i"} {
        return {{LRG 2}}
    }
    if {$via_name == "VIA5i"} {
        return {{SQR 1}}
    }
}
proc get_via_variations {via} {
    return "${via} ${via}BAR2"
}

"VIA4iLRG, VIA4iLRG2, VIA5i"
VIA_VARIATIONS "VIA4i, VIA4iBAR2, VIA5i, VIA5iBAR2"
```

As a result of the tool behavior, the VIA4iLRG2 layer must also be created; although, it is not listed in the via variations.

Calibre YieldEnhancer PowerVia Files and Specifications

The files and specifications used in the flow depend on your foundry process and may vary from the examples in this section. See your Siemens EDA representative for assistance with customizing your inputs for the Calibre YieldEnhancer PowerVia flow.

Via Variables File Specifications	201
Enclosure Template File Specifications	207
Generated Enclosure Template File Specifications	208
Calibre PERC Flow File Specifications	209
Backannotation Flow Specifications	210
Net Name Specifications	213
Calibre YieldEnhancer PowerVia techlib.tcl File Specifications	214

Via Variables File Specifications

The via variables file is a text file that describes the via parameters.

Only lines with the “VARIABLE” keyword are parsed in this file with leading whitespaces allowed. The VARIABLE keyword is followed by the name of the variable and the value. The value can be any real number. The file is case insensitive.

Mandatory for All Technology Nodes

The following parameters are mandatory for all technology nodes:

- **width** — The length of the short edge of the via.
- **length** — The length of the long edge of the via (same value as the width parameter for square vias).
- **step** — The distance in x- and y-directions between the fill vias (interpreted to be in the positive x- and y-direction).
- **offset** — The offset between the fill vias to allow for staggering (interpreted to be in the positive x- and negative y-direction).
- **enclosure** — The value used for fill area adjusting.
- **space** — The space from a different via variation of the same via or from an existing via and a same via that is inserted. See “[Spacing Between Vias](#)” on page 185 and “[Spacing from BAR Vias](#)” on page 189.

The variable names are specified as follows:

- *via_WIDTH* — Specifies the width for each via type (square, bar, large).
- *via_LENGTH* — Specifies the length for each via type (square, bar, large).
- *via_STEP* — Specifies the single step-size in both the x- and y-directions.

Note

 For square (SQR) and large (LRG) vias, the WIDTH and LENGTH should be equal.

Or:

- *via_STEPx* — Specifies the single step-size in the x-direction.
- *via_STEPy* — Specifies the single step-size in the y-direction.

Or for bar vias:

- *via_LONGEDGE_STEP* — Specifies the single step-size between the long edges of inserted bar vias.
- *via_SHORTEDGE_STEP* — Specifies the single step-size between the short edges of inserted bar vias.
- *via_OFFSET* — Species the fill offset value used for the via.
- *M_via_ENC* — Species the enclosure area used for the via.
- **SPACE** — Specifies the spacing distance for each pair of via types using the following naming convention:
 - *via_viaVariation_SPACE* — Specifies the space from a different via variation of the same via.
 - *via_via_SPACE* — Specifies the space from an existing via to a fill via of the same type.

Or for bar vias (see also “[Spacing from BAR Vias](#)” on page 189):

- *via_viaBAR_LONGEDGE_SPACE* — Specifies the space from the long edge of an existing bar via to a fill via (square, BAR, or LRG).
- *via_viaBAR_SHORTEDGE_SPACE* — Specifies the space from the short edge of an existing bar via to a fill via (square, BAR, or LRG).

For example:

```
VARIABLE via_via_SPACE distance_microns
VARIABLE via_viaBAR_SPACE distance_microns
VARIABLE via_viaLRG_SPACE distance_microns
VARIABLE viaBAR_viaLRG_SPACE distance_microns
```

Or:

```
VARIABLE via_via_SPACE distance_microns
VARIABLE via_viaBAR_SHORTEDGE_SPACE distance_microns
VARIABLE via_viaBAR_LONGEDGE_SPACE distance_microns
VARIABLE via_viaLRG_SPACE distance_microns
VARIABLE viaBAR_viaLRG_SPACE distance_microns
```

Optional for All Technology Nodes

The following variables are optional for all technology nodes:

- *via1_via2_SPACE* (for same-layer spacing).
- *via1_via2_LONGEDGE_SPACE* (for bar vias spacing).
- *via1_via2_SHORTEDGE_SPACE* (for bar vias spacing).
- *via_DIFFERENT_NET_SPACE*(see “[Connectivity-Based Spacing for Same-Layer Vias](#)” on page 192, also includes variables for SHORTEDGE and LONGEDGE different net spacing).
- *via1_via2_DIFFERENT_NET_SHORTEDGE_SPACE* (for bar vias spacing).
- *via1_via2_DIFFERENT_NET_LONGEDGE_SPACE* (for bar vias spacing).
- *via_COUNT* (see “[Via Count Rules Filtering](#)” on page 196).
- *via_STEPx, via_STEPy* (if not specified, then *via_STEP* must be specified).
- *via_LONGEDGE_STEP, via_SHORTEDGE_STEP* (for bar vias, if not specified, then *via_STEP* must be specified).
- *via_viaBAR_SHORTEDGE_SPACE, via_viaBAR_LONGEDGE_SPACE* (for bar vias, if not specified, then SPACE must be specified). See “[Spacing from BAR Vias](#)” on page 189.
- *UpperViaType_LowerViaType_SPACE* for Vx to Vx-1 spacing. See “[Spacing Between Vias](#)” on page 185.

Variables Specific to Foundry Process Flows

Note

 The via variables specifications for the following flows are specific to your foundry process. Depending on your technology flow, you use a predefined or a generated via variables file. If you have questions on the content of these files, configuration instructions are provided by your foundry or Siemens Digital Industries Software. Contact your Siemens EDA representative for details if needed.

(predefined) Via Variables File Process Flows

This flow uses a predefined via variables file that contains the via parameters for each metal group.

Mandatory for this flow:

For all via types (square, bar, and large) the sizes as specified by the width and length parameters are mandatory. For example:

```
VARIABLE V2_WIDTH 0.02
VARIABLE V2_LENGTH 0.02
VARIABLE V2BAR_WIDTH 0.02
VARIABLE V2BAR_LENGTH 0.02
VARIABLE V2LRG_WIDTH 0.02
VARIABLE V2LRG_LENGTH 0.02
...
```

These parameters are specified along with the variable statements for the step, offset, enclosure, and space parameter values. For example:

```
// V2 variables
VARIABLE M_V2_ENC 0.02
VARIABLE V2_WIDTH 0.02
VARIABLE V2_LENGTH 0.02
VARIABLE V2_STEPx 0.03
VARIABLE V2_STEPy 0.04
VARIABLE V2_offset 0.0
...
// V2-V2 space variables
VARIABLE V2_V2_SPACE 0.034
...
```

For certain technologies using Ax (A4, A5 ...) vias, the “*via_lower_metalfinger*” variable is required for square and LRG vias. This uses a specific method for filling vias in areas specified as “finger”. For example:

```
VARIABLE A4_C4finger 0.632
```

Optional for this flow:

For certain technologies using Ax (A4, A5 ...) vias, these step-size variables are optional.

Optional for square and large via types:

- *via_lower_metalfinger_STEPx*
- *via_lower_metalfinger_STEPy*

Optional for bar via types:

- *via_lower_metalfinger_LONGEDGE_STEP*

- *via_lower_metalfinger_SHORTEDGE_STEP*

For certain technologies using Ax (A4, A5 ...) vias, these space variables are optional.

- *via1_via2_lower_metalfinger_LONGEDGE_SPACE*
- *via1_via2_lower_metalfinger_SHORTEDGE_SPACE*

Where via1 and via2 can be type square, BAR, or LRG, and at least one of via1 and via2 is type BAR.

If the above two space variables are not specified, then the following space option is used:

via_lower_metalfinger_SPACE

Note

 The naming convention of the variable statement follows the insertion priority of the different via types, which are prioritized starting with the addition of large (LRG), followed by bar (BAR), and then square via types. In the variable statement, the lower priority via type is specified to the left of the higher priority via type.

For example:

```
VARIABLE A4_C4finger_STEPx 0.092
VARIABLE A4_C4finger_STEPy 0.064
VARIABLE A4BAR_C4finger_LONGEDGE_STEP 0.05
VARIABLE A4BAR_C4finger_SHORTEDGE_STEP 0.02
VARIABLE A4_A4BAR_C4finger_LONGEDGE_SPACE 0.04
VARIABLE A4_A4BAR_C4finger_SHORTEDGE_SPACE 0.08
VARIABLE A4BAR_A4LRG_C4finger_SHORTEDGE_SPACE 0.08
VARIABLE A4BAR_A4LRG_C4finger_LONGEDGE_SPACE 0.57
```

(generated) Via Variables File Process Flows

This flow has an intermediate step that generates a new via variables file during the run based on the information in the input via variables file. The generated via variables filename has the format *<filename>_gen*. For example, set the directory path (full or relative) in the *techlib.tcl* file:

```
set VIA_VARIABLES "/usr/project/work/var_types"
```

The input file with filename *var_types* must be located at the specified directory path and is used to generate a new via variables file with filename *var_types_gen* in the output directory. The input file contains information for each via type as specified in the BEOL layer stack. The generated via variables file contains the via parameters for each via from the layer stack.

For example, if VIAli has via type Vxa in the input via variables file:

```
VARIABLE Vxa_WIDTH 0.020
```

The line becomes the following in the output via variables file:

```
VARIABLE VIA1i_WIDTH 0.020
```

Mandatory for this flow:

For all via type variations (square, bar, and large), the sizes as specified by the width and length parameters are mandatory. For example:

```
VARIABLE Vx_WIDTH 0.020
VARIABLE Vx_LENGTH 0.020
VARIABLE VxBAR_WIDTH 0.020
VARIABLE VxBAR_LENGTH 0.040
VARIABLE VxLRG_WIDTH 0.080
VARIABLE VxLRG_LENGTH 0.080
...
...
```

For extended via types, the width and length parameters must also be specified:

```
VARIABLE VxaBAR2_WIDTH 0.020
VARIABLE VxaBAR2_LENGTH 0.040
```

The width and length parameters are specified along with the variable statements for the step, offset, enclosure, and space parameter values. For example:

```
// Vx variables
VARIABLE Mx_Vx_ENC 0.002
VARIABLE Vx_WIDTH 0.02
VARIABLE Vx_LENGTH 0.02
VARIABLE Vx_STEP 0.04
VARIABLE Vx_offset 0.0
...
// Vx-Vx space variables
VARIABLE Vx_Vx_SPACE 0.064
...
// Vx count variables (optional)
VARIABLE Vx_count 1
...
```

Optional for this flow:

For some via types, there can be two sizes for bar vias. In these cases, if the variable *via_LENGTH2* is added to the input via variables file, then the flow also extracts these bar via sizes. For example:

```
VARIABLE VxBAR_LENGTH 0.040
VARIABLE VxBAR_LENGTH2 0.060
```

When there are two length specifications for a particular bar via, both of these specifications are considered as one bar via layer. The count checks and spacing constraints are applied to both bar via specifications as one layer.

All other variables (VxBAR_COUNT, Vx_VxBAR_SPACE, ...) are the same for both sizes of bar vias (VxBAR_LENGTH, VxBAR_LENGTH2).

Enclosure Template File Specifications

The enclosure template file contains the via and metal width-dependent enclosure constraints.

Each line in the enclosure template file contains the via and metal names, width constraint, enclosure, and an optional extended value separated by one or more whitespaces (space or tab) in the following sequence:

- via name
- metal name
- “width” keyword followed by width_range value:
 - “less” keyword and value (width < value)
 - “more” keyword and value (width > value)
 - “lessOrequal” keyword and value (width <= value)
 - “moreOrequal” keyword and value (width >= value)
 - value1 value2 (width >= value1 and width < value2)
 - value1 (width == value1)
- “enc” keyword and values (two real numbers)

For BAR vias, the values are order-specific: the first value is for the long edge; the second value is for the short edge.

- “ext” keyword and value (one real number) (optional)

The line syntax format is as follows:

via_name metal_name width width_range enc values [ext value]

The file is case insensitive and may contain empty lines and comments (// comment). For example,

```
VIA0i M1i width less 0.058 enc 0.01 0.05 ext 0.0020 // M2_W2426
VIA7i M7i width 0.040 0.060 enc 0.01 0.03
VIA7iBAR M7i width 0.050 0.070 enc 0.02 0.01
VIA9i M9i width more 0.038 enc 0.0 0.05
// any comment here
...
```

For each via and metal pair, the width_range for the metal must not intersect with another width_range for the same via metal pair. For BAR vias, the minimum enclosure value is from the long edge of the inserted via, and the maximum value is from the short edge.

Generated Enclosure Template File Specifications

Certain foundry processes use an input file with via enclosure specifications to generate the via enclosure template file used for the Calibre YieldEnhancer PowerVia flow.

Note

 The via enclosure template files and content depend on your foundry process and may vary from the files and specifications listed. See your Siemens EDA representative for assistance with customizing your inputs for the Calibre YieldEnhancer PowerVia flow.

You customize the input file with specifications for the via and metal enclosure values used by your foundry process in a template provided by your Siemens EDA representative. See “[Enclosure Template File Specifications](#)” on page 207.

The path to the input filename is set by the following variable in your *techlib.tcl* file:

```
set ENC_TEMPLATE "<file_path>/enc_filename"
```

The input file, *enc_filename*, is used to generate the enclosure template file, *enc_filename_gen*, during the run.

The content of the input file includes information for each via type in your layer stack. The via and metal enclosure values in the input file are defined per via type (Vx, Vy, ...) for both lower- and upper-metal. Specifications for large, bar, and square vias are included. The input file may include enclosure rules for all possible via types, even if the via types are not used in the specified layer stack. See “[Calibre YieldEnhancer PowerVia techlib.tcl File Specifications](#)” on page 214 and “[powervia](#)” on page 226 for layer stack specifications and variables.

The generated enclosure template file contains all the necessary via parameters for each via type from the layer stack and is generated to the output directory of the flow. The via parameters are used by the Calibre YieldEnhancer PowerVia flow for metal width and enclosure constraints.

The following table shows the input and output lines for an enclosure template file for two vias of “Vxa” type, VIA1i and VIA2i.

Note

 For BAR vias, the order of the enclosure values after the “enc” keyword matters: the first value is for the long edge, and the second value is for the short edge, correspondingly.

Table 9-3. Example Enclosure Template File Lines

Input Enclosure Template File Lines	Generated Enclosure Template File Lines
Vx M_LOWER width 0.8 0.9 enc 0.18 0.03	VIA1i M1i width 0.8 0.9 enc 0.18 0.03 VIA2i M2i width 0.8 0.9 enc 0.18 0.03
VxBAR M_LOWER width 0.6 0.7 enc 0.17 0.2	VIA1iBAR M1i width 0.6 0.7 enc 0.17 0.2 VIA2iBAR M2i width 0.6 0.7 enc 0.17 0.2
VxLRG M_LOWER width 0.6 0.7 enc 0.17 0.2	VIA1iLRG M1i width 0.6 0.7 enc 0.17 0.2 VIA2iLRG M2i width 0.6 0.7 enc 0.17 0.2
Vx M_UPPER width 0.8 0.9 enc 0.18 0.03	VIA1i M2i width 0.8 0.9 enc 0.18 0.03 VIA2i M3i width 0.8 0.9 enc 0.18 0.03
VxBAR M_UPPER width 0.6 0.7 enc 0.17 0.2	VIA1iBAR M1i width 0.6 0.7 enc 0.17 0.2 VIA2iBAR M3i width 0.6 0.7 enc 0.17 0.2
VxLRG M_UPPER width 0.6 0.7 enc 0.17 0.2	VIA1iLRG M1i width 0.6 0.7 enc 0.17 0.2 VIA2iLRG M3i width 0.6 0.7 enc 0.17 0.2

The input file can also contain lines that do not need parsing, where the via is from the stack, and the metal name is not encoded. In this case, both the input file and the generated enclosure template file contain the same entries. For example, this line in the input file is unchanged in the corresponding line of the generated enclosure template file:

VIA1i M1i width 0.8 0.9 enc 0.18 0.03

Calibre PERC Flow File Specifications

The Calibre YieldEnhancer PowerVia PERC flow requires an SVRF file containing Calibre® PERC™ layer specifications.

If the PERC_FLOW variable and file are set in the *techlib.tcl* file (see “[Variables for powervia Utility](#)” on page 230), the Calibre YieldEnhancer PowerVia flow uses the PERC flow layers defined in the given PERC SVRF file instead of generating the layers.

The PERC flow for Calibre YieldEnhancer PowerVia has the following file specifications:

- For each metal specified by the BEOL_STACK variable, the metal layers are defined with the following naming convention:

pg_{metal}

For example:

```
pg_M1 = COPY M1_SEL
pg_M2 = ...
pg_M3 = ...
pg_C4 = ...
pg_C5 = ...
pg_C6 = ...
```

- For each selected via (see SELECT_LAYER in “[Variables for powervia Utility](#)” on page 230), the fill area layers have the following naming convention:

{lower_metal}_{upper_metal}_fillarea_pre

For example, if V2 and AM vias are selected from the BEOL stack, then the lower and upper layers are defined respectively, as follows:

V2 via — M2, M3

AM via — M3, C4

For example:

```
M2_M3_fillarea_pre = COPY M2_M3_fillarea_PERC M3_C4_fillarea_pre = ...
```

Backannotation Flow Specifications

The via backannotation flow provides the capability to output vias generated by the Calibre YieldEnhancer PowerVia flow in incremental Design Exchange Format (DEF).

The generated incremental DEF file is an ASCII editable file that can be used to backannotate the added via enhancements into your original design database using specific syntax.

To run the backannotation flow, set the required and optional variables and execute the [powervia](#) command line arguments. This generates an incremental DEF file in the output directory of the Calibre YieldEnhancer PowerVia run. The incremental DEF file can then be imported with the original design database in a layout-viewer tool. Further analysis can be done on the database with the inserted vias, such as timing and power analysis.

The following is the list of variables and Tcl procedures used in the backannotation flow.

Note

 See “[powervia Variables](#)” on page 230 and “[Calibre YieldEnhancer PowerVia techlib.tcl File Specifications](#)” on page 214 for complete descriptions and usage information for the variables and Tcl procedures used in the Calibre YieldEnhancer PowerVia flow.

Variables used in the backannotation flow:

- RUN_DEF_BA *flag* — (Required) Enables the backannotation flow (YES or NO).

- DEF DESIGN *design_name* — (Optional) By default, the design name is set to *primary*, which is a cell name specified in LAYOUT PRIMARY.
- DEF FILE *file_name* — (Optional) Sets an incremental DEF file name. Default is *incremental_<primary>.def*.
- DEF VERSION *ver_num* — (Optional) Sets an incremental DEF version. Default is 5.8.

Tcl procedures used in the backannotation flow:

- proc via_name_in_DEF {via} {} — (Optional) Modifies a via name in the DEF.
- proc get_def_layer_name {layername} {} — (Optional) Enables setting of metal and via layer names in DEF.
- proc set_def_net_type {netname} — (Optional) Specifies a net type for the given net name.

The incremental DEF file conforms to standard DEF syntax. You can edit the statements in the file to create via and net updates for your design by using this syntax. For example, you can define a via cell in the “Vias” section of the file as “VIAS *number*”, where *number* is the count of the via cells in that section:

```
VERSION 5.8 ;
DESIGN TOPCELL ;
UNITS DISTANCE MICRONS 2000 ;
VIAS 1 ;
- YE_VIA1
+ RECT M1 ( -20 -20 ) ( 20 20 )
+ RECT VIA1 ( -20 -20 ) ( 20 20 )
+ RECT M2 ( -20 -20 ) ( 20 20 ) ;
END VIAS
```

The default name of the created via cell is YE_via. If the Tcl procedure “via_name_in_DEF” is defined, then the returned name from the procedure is used instead of YE_via.

RECT statements for via cell layers M1, VIA1, and M2 specify the lower-metal, via, and upper-metal layers, respectively. Terms enclosed in parenthesis $(-w/2 \text{ } -w/2) \text{ } (w/2 \text{ } w/2)$ define the center origin of the via cell, where w is the width of the via in database units.

After defining the via cell, you can reference it for placement to a specific coordinate location in the SPECIALNETS section of the incremental DEF file. The SPECIALNETS section defines the netlist connectivity and special routes for nets containing special pins. Special routes are normally used for power routing, fixed clock nets, and clock-mesh routing. For example, you

can specify to place the created via cell on net VDD in the SPECIALNETS section of the incremental DEF file:

```
SPECIALNETS 1 ;
- VDD
+ ROUTED M1 40 ( 1020 1020 ) YE_VIA1
+ USE POWER ;
END SPECIALNETS
```

Note

 A via cell must already be created in the VIAS section before it can be referenced in the SPECIALNETS section.

The SPECIALNETS *number* specifies the count of the nets in this section. For example, to add a via cell on net VDD, you specify “- VDD” and add the route on the lower-metal layer (ROUTED M1) followed by the route width (40 database units), which is the same as the width of the via cell. The placement coordinates for the via cell are midpoint on the net. Placement coordinates (x1 y1) specify the location of the via origin followed by the via cell name.

You can also use the SPECIALNETS section for specifying the net type (usage). For example, this syntax specifies VDD as a power net:

```
SPECIALNETS 1 ;
- VDD + ROUTED M1 40 ( 1020 1020 ) YE_VIA1
+ USE POWER ;
END SPECIALNETS
```

The valid values for net type are “ANALOG”, “CLOCK”, “GROUND”, “POWER”, “RESET”, “SCAN”, “SIGNAL”, and “TIEOFF”. If no net type is specified, the default is “SIGNAL”. Specifying an invalid net type generates an error and causes the tool to exit.

Alternatively, you can also set the net type by one of the following methods:

- In the *techlib.tcl* file Tcl procedure that sets the net type for DEF_BA in the backannotation flow:

```
proc set_def_net_type {netname}
```

- In the nets file specified by powervia command line options or a variable in the *techlib.tcl* file:

```
VDD POWER // type defined
CLK //type undefined
```

The Tcl procedure and the nets file option cannot both be specified; in this case, the Tcl procedure specifications are ignored. For example, the net type for CLK defaults to net type SIGNAL, because the nets file takes precedence:

Nets file:

```
VDD POWER
CLK
```

Tcl proc:

```
proc set_def_net_type {netname} {
    if {$netname == "CLK"} {return "CLOCK"}
}
```

Net Name Specifications

The Calibre YieldEnhancer PowerVia flow uses an internal case-sensitive mode for proper handling of duplicate net name specifications.

User-specified nets with matching names, differing only by case (for example, “VDD” and “vdd”) are renamed internally by the flow for processing. This case-sensitive mode of the flow applies to net names in files specified by the environment variable `POWER_VIA_NETS_FILE_NAME`, and net names specified by the `techlib.tcl` file “NETS” variable, and other net file variables. Refer to “[Variables for powervia Utility](#)” on page 230 for a list of variables that are specific to net specifications in the `techlib.tcl` file.

Note

 If your process design kit contains “VIRTUAL CONNECT NAME *name* [*name...*]” statements, you must add the “Layout Preserve Case Yes” statement if you want to use case-sensitivity for net names.

For reference, if net names with matching criteria are specified, the original and the modified net names are mapped during the flow into the `net_names.map` file. The following table shows an example of the original and the mapped net names in the `net_names.map` file.

Table 9-4. Net Names Mapping

Original Net Name	Mapped Net Name
VSSA	VSSA_case1
vSSa	vSSa_case0
VCCA_CORE	VCCA_CORE
vssa	vssa

The format of the net names in the file is such that the original specification of the net name is not modified, unless a case-sensitive match exists. The net names with case variations are renamed only for internal processing during the flow.

Calibre YieldEnhancer PowerVia techlib.tcl File Specifications

The *techlib.tcl* file contains configuration information for running the Calibre YieldEnhancer PowerVia flow.

The content of this file defines variable settings, Tcl procedures, and user controls for the flow.

Note

 Your *techlib.tcl* file content depends on your foundry process and node and may vary from the content listed. See your Siemens EDA representative for assistance with customizing your inputs for the Calibre YieldEnhancer PowerVia flow.

techlib.tcl Variables

Refer to “[Variables for powervia Utility](#)” on page 230 for a list of variables that are specific to the *techlib.tcl* file.

techlib.tcl Procedures

The following Tcl procedures are common for all technology nodes. Additional *techlib.tcl* file procedures may be used for your specific foundry flow.

`proc assign_via_insertion_order {via_name} {}`

This Tcl procedure specifies the order of insertion of via types defined by *via_name*. It returns either an empty string or the selected via types for the specified *via_name*. The flow maintains the fill order of the vias as specified by the procedure.

The following behavior applies:

- If the procedure is not defined, then the flow uses the default order for inserting the via types (LRG > BAR > SQR), and if defined for only certain vias, the default order is used for the remaining via types.
- If via ordering is defined for a certain via, it must contain all sizes of this via, otherwise, the flow exits with an error message.
- If a via in the return list is *not* in the selected via types for the defined via, then a warning message is printed, and the via is not inserted.

For example:

```
proc assign_via_insertion_order {via_name} {
    if {$via_name == VIA13 || $via_name == VIA14} {
        return "${via_name}BAR ${via_name}BAR2 ${via_name}LRG ${via_name}"
    } else return ""
}
```

If the BAR via area is greater than the LRG via area, you can specify the following order “*via_nameBAR via_nameLRG*” to insert the larger via first for optimal insertion:

If the area of the higher-ordered via is less than the area of the lower-ordered via, a warning message is printed, and the flow continues normally.

proc net_property_layer {stackLayer} {}

This Tcl procedure enables net names to be defined as properties. The procedure returns the original layer name followed by a property name (for OASIS designs) or a property number (for GDS designs). The procedure uses the following syntax:

```
proc net_property_layer {stackLayer} {
    return "layer property_name | property_number"
}
```

For example, the procedure returns the following for a GDS format design:

```
proc net_property_layer {stackLayer} {
    return "$stackLayer 10"
}
```

proc extended_type_count {via_name} {}

This Tcl procedure works in conjunction with the SELECT_LAYER variable to return a list containing the count of extended via types (BAR, BAR2, ... LRG, LRG2, ...). If this Tcl procedure is specified, the VIA_TYPE variable is ignored.

The return list for the procedure is specified as follows:

- return { {LGR *n*} {BAR *m*} {SQR *q*} }
- where the returned via counts (*n*, *m*, and *q*) can be zero or a positive integer.
- *n* — Count of LRG via type
 - *m* — Count of BAR via type
 - *q* — Count of SQR via type (either 1 or 0 are valid)

The procedure works with SELECT_LAYER as follows:

- If via_name is defined in SELECT_LAYER, but all the counts of extended via types are zero, the tool generates an error.

- If via_name is *not* defined in SELECT_LAYER, and all the counts of extended via types are zero, the tool continues normally, not inserting that via_name.
- If via_name is defined in SELECT_LAYER, but has no count entries of extended via types, the tool inserts SQR vias as per VIA_TYPE.

For example:

```

set SELECT_LAYER "SINGLE"
proc extended_type_count {via_name} {
    if {$via_name == VIA1i || $via_name == VIA2i} {
        return { {LGR 5} {BAR 2} {SQR 1} }
    }
}
# Means 1 SQR, 2 BAR, 5 LRG for VIA1i and VIA2i need to be inserted.
# SQR: VIA1i, VIA2i
# BAR: VIA1iBAR, VIA1iBAR2, VIA2iBAR, VIA2iBAR2
# LRG: VIA1iLRG, VIA1iLRG2, VIA1iLRG3, VIA1iLRG4, VIA1iLRG5, \
VIA2iLRG, VIA2iLRG2, VIA2iLRG3, VIA2iLRG4, VIA2iLRG5

set SELECT_LAYER "V1 V2 V5 V6 V7"
proc extended_type_count {via_name} {
    if {$via_name == V1 || $via_name == V2} {
        return { {BAR 2} {LGR 0} {SQR 0} }
        # Means 2 BAR: V1BAR, V1BAR2, V2BAR, V2BAR2
    } elseif {$via_name == V5} {
        return { {LGR 5} {BAR 4} {SQR 1} }
        # Means 1 square, 4 BAR, 5 LRG
    } elseif {$via_name == V6} {
        return { {BAR 0} {LGR 0} {SQR 1} }
        # Means neither LRG, nor BAR - only SQUARE
    } else {
        return { {SQR 0} {LGR 5} {BAR 0} }
        # Means only 5 LRG
    }
}

set SELECT_LAYER "V1 V2"
proc extended_type_count {via_name} {
    if {$via_name == V1} {
        return { {BAR 0} {LGR 0} {SQR 0} }
    } -> "ERROR: All types of selected via $via cannot be 0."
}

```

The procedure must return valid values for all specified layer names in the stack; returning an empty string results in an error. See “[Original Via Layers Extraction for Extended Via Types](#)” on page 198 for an example using the generated via variables technology flow.

proc get_def_layer_name {layername} {}

This Tcl procedure enables you to set metal and via layer names in the DEF file. This is useful when some via and metal layer names in the place and route flow are different from those

defined in the layer stack and provided as input to the Calibre YieldEnhancer PowerVia flow.
For example:

```
proc get_def_layer_name {layername} {
    if {$layername == "VIA1i"} {return "VIA1" }
    if {$layername == "M1i"} {return "metall1" }
    return $layername}
```

If this procedure is defined, it must return valid values for all specified layer names in the stack, returning an empty string results in an error.

proc get_via_variations {via} {}

This Tcl procedure returns a string that contains all variations of via types in the original design for a particular via layer. For example:

```
proc get_via_variations {via} {
    return "$via ${via}BAR ${via}LRG"
}
```

proc get_explicit_stack {}

This Tcl procedure uses the layer stack name defined by the BEOL_STACK variable to return a string listing all the layers in the BEOL stack. For example, (M1 - V1 - M2 - V2 -... - M10).

proc count_for_wide_metal {via} {}

This Tcl procedure is called per via type and is used to specify via count constraints. Each via type has a different minimum count per metal width and must be specified separately. For example:

```
proc count_for_wide_metal {via} {
    switch -exact $via {
        VIA1 {return [ list \
            "{0 1000 2}" \
            "{0 1000 2}" ] }
        VIA1BAR { return [ list \
            "{0.12 1000 2}" \
            "{0.12 1000 2}" ] }
    }
}
```

proc get_exclusion_area {layerName} {}

This Tcl procedure returns the definition of an exclusion area for the input via layer name. For example:

```
proc get_exclusion_area {layername} {
    if {$layername == "VIA0"} {return "VIA0_general_exclude_layer"}
    if {$layername == "VIA0BAR"} \
        {return "VIA0BAR_specific_exclude_layer"}
}
```

- VIA0_general_exclude_layer is applied as an exclude layer for all VIA0 types.
- VIA0BAR_specific_exclude_layer is applied as an exclude layer for only VIA0BAR types.

If the exclusion area is not defined for the input layer, the procedure returns an empty string. See “[Exclusion Areas](#)” on page 195 for more examples.

proc alter_group_fill_region {via fill_region} {}

This Tcl procedure derives a new layer to be used as a modified fill region.

The inputs are as follows:

- *via* — The via name for which the fill region is to be modified.
- *fill_region* — The fill area generated by the flow.

The procedure returns either an empty string, meaning that no altering occurred, or a new layer with the name FILL_REGION_CORRECT_ *via*. This derived layer is used as a modified fill region for the *via*. The layer derivation is included in the return string. For example:

```
proc alter_group_fill_region {via fill_region} {
    set via_first_letter [string index $via 0]
    if {$via_first_letter != "C"} {return}
    return "region_extent$via = extents $FILL_REGION
        FILL_REGION_filter1_$via = DFM FILL stripe_V1$via
        ...
        FILL_REGION_CORRECT_$via = ..."
}
```

proc alter_inserted_vias {original_via inserted_via fill_region lower_metal upper_metal} {}

This Tcl procedure derives a new layer for the output inserted via layer.

The inputs are as follows:

- *original_via* — The via name for which the output via is to be modified.
- *inserted_via* — The inserted via name (for example: *viaCFILLR_final*).
- *fill_region* — The fill area generated by the flow.
- *lower_metal* — The lower metal layer of the via.
- *upper_metal* — The upper metal layer of the via.

The procedure returns either an empty string, meaning that no altering occurred, or a new layer with the name *original_via_CORRECT*. This derived layer is used to output the inserted via layer. The layer derivation is included in the return string. For example:

```
proc alter_inserted_vias {original_via inserted_via fill_region \
    lower_metal upper_metal} {
    if {$original_via == "VIA2" || $original_via == "VIA2BAR" || \
        $original_via == "VIA5"} {
        return "wide_upper_metal$via_name = $upper_metal WITH WIDTH > 0.02
${via_name}_CORRECT = $inserted_via NOT wide_upper_metal$via_name"
    }
    return ""
}
```

proc exclude_ip_cells {via_name} {}

This Tcl procedure excludes via insertion from certain IP cells. The specified via_name applies to all types for that via. For example, if the via_name is “Via1i”, then the cell exclusion is for all types (Via1i, Via1iBAR, Via1iLRG) of the selected vias controlled by the SELECT_LAYER variable. See “[Insertion of Different Via Types \(LRG, BAR, SQUARE\)](#)” on page 183 for information on the variable specification.

Of the two items returned by the procedure, the first is either an empty string (no IP cells to exclude) or a space-separated list that contains the IP cell names to exclude from the via insertion. The second (and optional) item is a nonnegative numeric value, by which the cell shapes on the specified fill layers are sized down. For example:

```
proc exclude_ip_cells {via_name} {
    if {($via_name == "VIA0i") } {return {{*ip1*} {0}} }
    if {($via_name == "VIA1i" )} {return {*ip1* *ip2*} }
    if {($via_name == "VIA2i" )} {return "*ip1*" }
    return ""
}
```

Regular expressions are supported in the cell name specifications, including one or more wildcard characters (*) for name matching. Cell names with wildcard characters must be enclosed in quotes or grouped by braces ({}); otherwise, a compilation error results, because the * is a reserved character in Tcl. The standard conventions for Tcl and regular expressions apply to the procedure specifications.

The procedure excludes the fill area for each of the specified vias. If a numeric value is returned, the shapes of the selected cells in the list are sized down by the given numeric value and then excluded from the layer where the via fill is performed. IP block layers for the vias are derived based on the procedure and are created with layer name “*via_IP_block_rule*”. These layers are output to the debug database (*net_aware_pwr_gnd.oas*) in the directory specified by the DEBUG_DIR variable. If the DEBUG_DIR variable is not specified, then the database is written to the current working directory.

proc configure_rdb_results {via_layer} {}

The parameter *via_layer* is the via layer name.

This procedure enables results database (RDB) file generation. Based on the return value, it configures the corresponding output layers to the RDB file. The output layers can be used for reviewing areas of interest in the layout and determining which vias need to be added in the flow. The RDB_FILE *techlib.tcl* variable sets the name of the RDB file. See “[powervia Variables](#)” on page 230.

The procedure returns a list of *{layerSpecifier netOption}* pairs. The valid values for each field are as follows:

- *layerSpecifier* — Specifies the layer names to output to the RDB file. See [Table 9-5](#) on page 220.
- *netOption* — Specifies keywords for_all, per_net, or specific_net_name.
 - for_all — Specifies only one layer, without net separation.
 - per_net — Specifies separate layers for each net that exists in the original design.
 - specific_net_name — Specifies only one layer for a net that exists in the original design (“VDD”, “vssa”, ...).

For example:

```
proc configure_rdb_results {via_layer} {
    if {$via_layer == "VIA0i" || $via_layer == "VIA10i"} {
        return "{$Mlower_Mupper_intersection per_net}
                  {$Mlower_Mupper_intersection_orig_improved for_all}
                  {$Mlower_Mupper_intersection VDD}"
    }

    if {$via_layer == "VIA3i"} {
        return "{$Via_inserted per_net}
                  {$Mlower_Mupper_intersection_miss_orig for_all}
                  {$Via_filtered_custom vssa}
                  {$Via_filtered_count for_all}"
    }
    ...
}
```

The following table shows the configuration parameters used for selecting the layers to output to the RDB file.

Table 9-5. RDB Configuration Parameters — Metal Intersection

Layer Specifier	Metal Intersection
Mlower_Mupper_intersection	Intersections for each pair of metal layers

Table 9-5. RDB Configuration Parameters — Metal Intersection (cont.)

Layer Specifier	Metal Intersection
Mlower_Mupper_intersection_miss_orig	Intersections with no original vias
Mlower_Mupper_intersection_miss_orig_miss_new	Intersections with no original vias/no inserted vias
Mlower_Mupper_intersection_miss_orig_improved	Intersections with no original vias/new inserted vias
Mlower_Mupper_intersection_orig_improved	Intersections with original/inserted vias

Table 9-6. RDB Configuration Parameters — Original/Inserted Vias

Layer Specifier	Original/Inserted Vias
Via_orig	Original vias
Via_inserted	Inserted vias
Via_filtered_enc	Filtered and inserted vias due to enclosure
Via_filtered_count	Filtered and inserted vias due to count
Via_filtered_custom	Filtered and inserted vias due to alter_inserted vias procedure

In the following example, there are two nets in the net list: “vssa” and “VCCA_CORE” specified in the configure_rdb_results procedure:

```
proc configure_rdb_results {via_layer} {
    if {$via_layer == "VIA0i"} {
        return "{$Mlower_Mupper_intersection
{$Mlower_Mupper_intersection_miss_orig
{$Mlower_Mupper_intersection_miss_orig_miss_new
{$Mlower_Mupper_intersection_miss_orig_improved
{$Mlower_Mupper_intersection_orig_improved
{Via_orig
{Via_inserted
{Via_filtered_enc
{Via_filtered_count
{Via_filtered_custom
for_all}
per_net}
vssa}
for_all}
VCCA_CORE}
for_all}
vssa}
for_all}
for_all}
per_net}"
    }
}
```

These are the SVRF code examples for the generated layers based on the return values for the configure_rdb_results procedure in the prior example:

- Intersections for each pair of metal layers:

```
// net-option "for_all"
M0i_M1i_intersection = OR "M0i_M1i_intersection_vssa"
"VIA0i_INSERTED_FINAL"
```

- Intersection of lower and upper metal layers with missing original vias:

```
// net-option "per_net"
M0i_M1i_intersection_miss_orig = (M0i_M1i_intersection NOT INTERACT
VIA0i) INTERACT "vssa_M0i"
M0i_M1i_intersection_miss_orig = (M0i_M1i_intersection NOT INTERACT
VIA0i) INTERACT "VCCA_CORE_M0i"
```

- Intersection of lower and upper metal layers with missing original vias and no inserted vias:

```
// net-option "vssa"
M0i_M1i_intersection_VCCA_CORE_miss_orig_miss_new =
((M0i_M1i_intersection NOT INTERACT VIA0i)
NOT INTERACT VIA0i_inserted) INTERACT "vssa_M0i"
```

- Intersection of lower and upper metal layers with missing original vias and new inserted vias:

```
// net-option "for_all"
M0i_M1i_intersection_miss_orig_improved =
M0i_M1i_intersection_miss_orig INTERACT
VIA0i_inserted_final
```

- Intersection of lower and upper metal layers with original and inserted vias:

```
// net-option "VCCA_CORE"
M0i_M1i_intersection_VCCA_CORE_orig_improved =
((M0i_M1i_intersection INTERACT VIA0i) INTERACT
VIA0i_inserted_final) INTERACT "VCCA_CORE_M0i"
```

- Original vias:

```
// net-option "for_all"
VIA0i_orig = COPY VIA0i
```

- Inserted vias:

```
// net-option "vssa"
VIA0i_inserted_final =
(OR VIA0iLRGCFILLR_final VIA0iBARCFILLR_final
VIA0iCFILLR_final) INTERACT "vssa_M0i"
```

The final inserted via layers are derived from the argument list of OR operations. Based on the return value of the *techlib.tcl* procedure, the layers can be generated in different ways. For example:

```
VIA0i_inserted_final =
(OR VIA0iLRCFILLR_final VIA0iBAR_CORRECT VIA0i_CORRECT)
INTERACT "vssa_M0i"
VIA0i_inserted_final =
(OR VIA0iLRG_CORRECT VIA0iBARCFILLR_final VIA0i_CORRECT) INTERACT
"vssa_M0i"
VIA0i_inserted_final =
(OR VIA0iLRG_CORRECT VIA0iBARCFILLR_final VIA0iCFILLR_final)
INTERACT "vssa_M0i"
```

- Vias inserted and filtered due to enclosure:

```
// net-option "for_all"
VIA1i_filtered_enc =
VIA1iCFILLR INTERACT VIA1i_region_bad
VIA1iLRG_filtered_enc =
VIA1iLRCFILLR INTERACT VIA1iLRG_region_bad
VIA1iBAR_vertical_filtered_enc =
VIA1iBARCFILLR_vertical INTERACT VIA1iBAR_region_bad_vertical
VIA1iBAR_horizontal_filtered_enc =
VIA1iBARCFILLR_horizontal INTERACT VIA1iBAR_region_bad_horizontal
VIA1iBAR_horizontal_refill_filtered_enc =
VIA1iBARCFILLR_horizontal_refill INTERACT
VIA1iBAR_region_bad_horizontal_refill
```

Where parameters are defined as follows:

- *viaCFILLR_orientation* — Initially inserted via layers.
- *via_region_bad_orientation* — Layers created due to enclosure filter.
- Vias inserted and filtered due to count:

```
// net-option "for_all"
"VIA0i_filtered_count =
(OR VIA0iCFILLR_remaining1 VIA0iCFILLR_remaining2) INTERACT
VIA0i_bad_count
VIA0iLRG_filtered_count =
(OR VIA0iLRCFILLR_remaining1 VIA0iLRCFILLR_remaining2) INTERACT
VIA0iLRG_bad_count
VIA0iBAR_vertical_filtered_count =
(OR VIA0iBARCFILLR_remaining1_vertical
VIA0iBARCFILLR_remaining2_vertical) INTERACT
VIA0iBAR_bad_count_vertical
VIA0iBAR_horizontal_filtered_count =
(OR VIA0iBARCFILLR_remaining1_horizontal
VIA0iBARCFILLR_remaining2_horizontal) INTERACT
VIA0iBAR_bad_count_horizontal
VIA0iBAR_horizontal_refill_filtered_count =
(OR VIA0iBARCFILLR_remaining1_horizontal_refill
VIA0iBARCFILLR_remaining2_horizontal_refill) INTERACT
VIA0iBAR_bad_count_horizontal_refill
```

Where parameters are defined as follows:

- *via_remaining1_orientation via_remaining2_orientation* — Inserted vias after count filter.
- *via_bad_count_orientation* — Layers created due to count filter.
- Vias inserted and filtered due to alter_inserted_vias procedure:

```
// net-option "per_net"
VIA0i_vssa_filtered_custom =
  ((OR VIA0iLRGCFILLR_final VIA0iBARCFILLR_final VIA0iCFILLR_final)
   NOT (OR VIA0iLRG_CORRECT VIA0iBAR_CORRECT VIA0i_CORRECT))
  INTERACT "vssa_M0i"

VIA0i_VCCA_CORE_filtered_custom =
  ((OR VIA0iLRGCFILLR_final VIA0iBARCFILLR_final VIA0iCFILLR_final)
   NOT (OR VIA0iLRG_CORRECT VIA0iBAR_CORRECT VIA0i_CORRECT))
  INTERACT "VCCA_CORE_M0i"
```

The argument list of OR operations generates the layers based on the user configuration (see proc alter_inserted_vias).

All layers names written to the RDB file have the prefix “Rule_” to distinguish them from existing layers names in the output database. For example:

M1_M2_intersection — Layer representation in the output database.

Rule_M1_M2_intersection — Layer representation in the RDB file.

proc set_def_net_type {netname}

This Tcl procedure specifies a net type for the given net name in the DEF file. For example:

```
proc set_def_net_type {netname} {
  if {$netname == "CLK"} {return "CLOCK"}
}
```

If the net name exists in both the Tcl procedure and the nets file, the nets file specification takes precedence.

proc via_name_in_DEF {via} {}

This Tcl procedure specifies any via name in the DEF file that you want to modify. For all selected vias in the flow, the procedure takes an argument via name and returns the name, which must exist in the DEF file. If a selected via is not defined in the DEF file or is an empty string, then the flow exits with an error message.

Calibre YieldEnhancer PowerVia Command Reference

The powervia utility along with certain commands, variables, and specifications are used for running the Calibre YieldEnhancer PowerVia flow.

powervia	226
powervia Variables	230

powervia

Adds additional vias to specified nets in a design and outputs an updated via database in GDSII or OASIS format.

Usage

Syntax 1

powervia

```
{ {[-turbo [thread_number]] [-hyper] [-remote host[,host ...]]]  
  [-remotefile config_file_name]} | [-cal_options calibre_options]}
```

Syntax 2

powervia

```
{-layout_path layout_path} {-beol_stack beol_stack_name}  
  {-nets_file_name nets_file_name} | {-nets_name nets_name}  
  { {[-turbo [thread_number]] [-hyper] [-remote host[,host ...]]]  
    [-remotefile config_file_name]} | [-cal_options calibre_options]}
```

Syntax 3

powervia ...a subset of arguments is specified with environment variables or *techlib.tcl* variables and others might not be.

Arguments

- **-layout_path layout_path**

Required argument that specifies the input layout design database. You can specify a GDSII or OASIS database. Both full and relative (./) paths (to current directory) are supported.

If not specified with command line syntax (Syntax 2), this argument must be specified with either the environment variable POWER_VIA_LAYOUT_PATH or the *techlib.tcl* variable LAYOUT_PATH (Syntax 3 or Syntax 1, respectively).

- **-beol_stack beol_stack_name**

Required argument that specifies the BEOL stack of your design.

If not specified with command line syntax (Syntax 2), this argument must be specified with either the environment variable POWER_VIA_BEOL_STACK or the *techlib.tcl* variable BEOL_STACK (Syntax 3 or Syntax 1, respectively).

- **-nets_file_name nets_file_name**

Required argument that specifies either the full or relative (./) path (to current directory) to the file that contains the net names for via insertion or to the name of this file in your working directory.

If not specified with command line syntax (Syntax 2), this argument must be specified with either the environment variable POWER_VIA_NETS_FILE_NAME or the *techlib.tcl* variables NETS_FILE_NAME or NETS (Syntax 3 or Syntax 1, respectively).

- **-nets_name *nets_name***

Required argument that specifies either the full or relative (./) path (to current directory) to the file that contains the net names for via insertion or to the name of this file in your working directory.

If not specified with command line syntax (Syntax 2), this argument must be specified with either the environment variable POWER_VIA_NETS_NAME or the *techlib.tcl* variables NETS_NAME or NETS (Syntax 3 or Syntax 1, respectively).

Note

 The functionality of the -nets_name argument is equivalent to that of the -nets_file_name argument. The -nets_name argument and its related variables are planned for deprecation in a future release.

- **-turbo [*thread_number*]**

Optional argument set that triggers multi-threaded parallel processing. The *thread_number* parameter is a positive integer that specifies the number of CPUs to use. By default, all available CPUs for which there are licenses are used (Syntax 1 and Syntax 2, only).

- **-hyper**

Optional keyword that enables hyperscaling mode. Hyperscaling enables the concurrent, parallel execution of rule file operations that maximizes CPU usage efficiency. This option must be specified with -turbo. It is generally not recommended to use this option with fewer than 8 CPUs (Syntax 1 and Syntax 2, only).

- **-remote *host[,host...]***

Optional keyword for Calibre MTflex run. At least one host parameter must be specified. The list of hostnames is comma-delimited and specifies that multiple hosts participate in multithreaded operations.

- **-remotefile *config_file_name***

Optional keyword for Calibre MTflex run. The *config_file_name* specifies the pathname of a configuration file containing information for the local and remote hosts.

- **-cal_options *calibre_options***

Optional keyword that specifies the command-line options for the Calibre nmDRC-H run. If specified, all the Calibre nmDRC-H command-line options for the run must be specified *after* the -cal_options keyword. If the -cal_options keyword option is not specified, you can specify the Calibre nmDRC-H command-line options individually as shown in Syntax 1 and Syntax 2.

Note

 You can only use one of these methods to pass the command-line options to the Calibre nmDRC-H run.

Description

Note

 A library of Tcl procedures and setup files is necessary for running the Calibre YieldEnhancer powervia utility. For details and content of these procedures, contact your Siemens EDA representative.

The powervia utility adds additional vias wherever possible to a DRC- and LVS-clean design. The via insertion is done DRC-clean, and the generated GDSII or OASIS output via database can then be merged with the input layout database and validated in a full Calibre nmDRC-H run.

The POWER_VIA_TECH environment variable and certain *techlib.tcl* file variables are required to run the powervia utility. The specification of the *techlib.tcl* variables, in some cases, depends on your foundry process technology. See “[Variables for powervia Utility](#)” on page 230 for a list of powervia variables.

There are three possible syntax formats that affect the control of the utility.

- **Syntax 1** — Use this syntax to run the complete powervia flow without making any modifications. The *techlib.tcl* file must contain all specifications and variables for your design and exist in your working directory.
- **Syntax 2** — Use this syntax to have finer control of inputs. Since this syntax uses command line options for all or a subset of the inputs, there is no order dependency for the options. If the options are not defined using the command line, then by default, they are picked up from the *techlib.tcl* file.
- **Syntax 3** — Specify all or part of the arguments by setting environment variables. If an option is specified that serves the same purpose as an environment variable, the variable setting is ignored. Similar to Syntax 2, use this syntax to have finer control of the inputs.

For command line specifications, see “[Running the powervia Utility](#)” on page 179 and the examples at the end of this section.

The following guidelines apply when using the powervia utility:

- The input layout database must be free of power and ground shorts and be LVS- and DRC-clean.
- The utility works only with annotated (text) input layout database designs.
- The via insertion can be done on any specified nets; however, the best results are obtained by the insertion of vias in power nets or wide bus nets.
- Inductors or capacitors must be identified using “blockage” layers. There is a specific blockage layer per via defined within the flow.

Examples

These are some examples of the specifications for running powervia.

You must specify the directory path to the *techlib.tcl* file and the path (default) to the required input files by setting the following environment variable in the command line *before* running the powervia utility.

```
setenv POWER_VIA_TECH <path>
```

Example 1

This is an example of Syntax 1, where the all inputs are located in the working directory:

```
setenv POWER_VIA_TECH /usr/project/work  
$MGC_HOME/bin/powervia -turbo
```

Example 2

This is an example of Syntax 2, where inputs are specified by command line options. All the omitted options must be set with environment variables or in the *techlib.tcl* file. If full paths or relative paths (to the current directory) are not specified, then files must be located in the working directory.

```
setenv POWER_VIA_TECH /usr/project/work  
$MGC_HOME/bin/powervia -turbo \  
-layout_path "/usr/project/design/design_in.oas" \  
-beol_stack 2X6YZa -nets_name pv_nets_file
```

Example 3

This is an example of Syntax 3, where environment variables and command line options are used.

```
// environment variable settings  
setenv POWER_VIA_TECH /usr/project/work  
setenv POWER_VIA_NETS_FILE_NAME /usr/project/design/inputs/pv_nets_file  
setenv BEOL_STACK 2X6YZa  
  
// powervia utility invocation  
powervia -turbo -layout_path "/usr/project/design/design_in.oas"
```

Example 4

This is an example of Syntax 1 or Syntax 2, where the Calibre nmDRC-H command-line options are passed using an expandable format after the *-cal_options* keyword:

```
setenv POWER_VIA_TECH /usr/project/work  
$MGC_HOME/bin/powervia -layout_path my_pv_input.gds \  
-beol_stack "my_pv_beol_stack" \  
-nets_name my_netnames.list \  
-cal_options -turbo -hyper remote -remote localhost
```

powervia Variables

The powervia utility uses certain variable specifications for running the Calibre YieldEnhancer PowerVia flow.

The applicable environment variables and *techlib.tcl* file variables are shown in the following table.

Table 9-7. Variables for powervia Utility

Variable Name	Description
POWER_VIA_TECH	 Note: (Required.) You must specify this environment variable in the command line <i>before</i> running the powervia utility. Set <i>value</i> to the path where the <i>techlib.tcl</i> file resides. Both full and relative (./) paths (to current directory) are supported.
RUN_DEF_BA	In <i>techlib.tcl</i> file: Set <i>value</i> to “YES” (case insensitive) to enable the backannotation flow. See “ Backannotation Flow Specifications ” on page 210.
DEF DESIGN	In <i>techlib.tcl</i> file: Set <i>value</i> to the design cell name of the DEF. By default, the design name is set to <i>primary</i> , the cell name specified in LAYOUT_PRIMARY. In the case of multiple designs, the first design topcell name is used. When DEF DESIGN is specified, the tool does <i>not</i> use Calibre DESIGNrev to retrieve the input database topcell. See “ Backannotation Flow Specifications ” on page 210.
DEF FILE	In <i>techlib.tcl</i> file: Set <i>value</i> to the filename of the incremental DEF file in the backannotation flow. If not specified, the default file name is <i>incremental_<primary>.def</i> , where <i>primary</i> is a cell name specified in LAYOUT_PRIMARY. See “ Backannotation Flow Specifications ” on page 210.

Table 9-7. Variables for powervia Utility (cont.)

Variable Name	Description
DEF_VERSION	<p>In <i>techlib.tcl</i> file:</p> <p>Set <i>value</i> to the DEF version number to use in the backannotation flow. By default, the DEF version is 5.8.</p> <p>See “Backannotation Flow Specifications” on page 210.</p>
INSERTED_VIA_FILE_PREFIX	<p>In <i>techlib.tcl</i> file:</p> <p>Set <i>value</i> to the filename of the output database with the filled vias.</p> <p> Note: Do not include the file extension (.oas/.gds), because the tool automatically appends the extension to the end of the filename based on OUTPUT_GDS.</p> <p>By default, the filename is <i>via_punch_fill.(oas/gds)</i>. If the <i>techlib.tcl</i> OUTPUT_DIR variable is not specified, the file is generated in the current directory; otherwise, it generates in the output directory.</p>
POWER_VIA_LAYOUT_PATH	<p>Environment variable:</p> <p>Specify in the command line where <i>value</i> is the full path or relative (./) path (to current directory) to the input layout database.</p>
LAYOUT_PATH	<p>In <i>techlib.tcl</i> file:</p> <p>Set <i>value</i> to full path of the input layout database including filename (.gds or .oas) using one of two options:</p> <ul style="list-style-type: none"> • Specify a full path enclosed in quotation marks (" ") . • Specify a relative path to your current run directory.
LAYOUT_PRECISION	<p>In <i>techlib.tcl</i> file:</p> <p>Set <i>value</i> to the precision value of the input design. Valid values are positive integers or floating-point numbers. If specified, the flow does not require a Calibre DESIGNrev license to retrieve the input database precision. Refer to “Calibre YieldEnhancer” in the <i>Calibre Administrator’s Guide</i> for complete licensing information.</p>

Table 9-7. Variables for powervia Utility (cont.)

Variable Name	Description
LAYOUT_SYSTEM	(Required.) In <i>techlib.tcl</i> file: Set <i>value</i> to “GDSII” or “OASIS” or “LEFDEF” for the format of the input layout database file.
MERGED_OUTPUT	This option is available only with LAYOUT_SYSTEM “GDSII” and “OASIS”. In <i>techlib.tcl</i> file: Set <i>value</i> to the full path and mode of the merged output. <ul style="list-style-type: none"> • <i>file_name</i> — Specify the full path, including file name, for the merged output of the original input and the <i>via_punch_fill.(oas/gds)</i> layout database files. • <i>mode</i> — Specify “append” or “rename”: <ul style="list-style-type: none"> • “append” — Concatenates the contents of the duplicate cells together. • “rename” — Renames all cells output from powervia to make them unique with added suffix “_powervia”.  Note: This option requires a Calibre DESIGNrev license. Refer to “ Calibre YieldEnhancer ” in the <i>Calibre Administrator’s Guide</i> for complete licensing information.
OUTPUT_DIR	In <i>techlib.tcl</i> file: Set <i>value</i> to control the path of the output files. If this environment variable is not specified, by default, the output files are written to your working directory. Both full and relative (./) paths (to current directory) are supported.
OUTPUT_GDS	(Required.) In <i>techlib.tcl</i> file: Set <i>value</i> to “Yes” for GDSII output layout database file.

Table 9-7. Variables for powervia Utility (cont.)

Variable Name	Description
SELECT_LAYER	(Required.) In <i>techlib.tcl</i> file: Set <i>value</i> to specify one of two modes of operation for via insertion: <ul style="list-style-type: none"> “SINGLE” — Inserts vias on <i>all</i> layers defined in the layer stack. “via1 via2 ... viaK” — Inserts vias <i>only</i> on the specified layers.
TECH_SPECIFIC_FLAG	(Required.) In <i>techlib.tcl</i> file: Set <i>value</i> to your foundry process technology keyword enclosed in quotation marks (“ ”).
ENC_TEMPLATE	(Required.) In <i>techlib.tcl</i> file: Set <i>value</i> to either the full or relative (./) path (to the current directory) that includes the file name of the via enclosure template file. This file is an editable template provided by your Siemens EDA representative and is required input for all foundry processes.
PERC_FLOW	In <i>techlib.tcl</i> file: Set <i>value</i> to absolute or relative (to the current directory) path to an SVRF file that contains the layers used for the Calibre® PERC™ flow. If this variable is set, the <i>techlib.tcl</i> flow uses the PERC flow layers defined in the specified file instead of generating these layers. See “ Calibre PERC Flow File Specifications ” on page 209. Refer to the <i>Calibre PERC User’s Manual</i> for more information on the Calibre PERC flow.
POWER_VIA_NETS_FILE_NAME	Environment variable: Specify in the command line where <i>value</i> is either the full path or relative (./) path (to the (current directory) to the file that contains net names for via insertion or to the name of the net names file in your working directory. This variable performs the same functionality as POWER_VIA_NETS_NAME.

Table 9-7. Variables for powervia Utility (cont.)

Variable Name	Description
NETS_FILE_NAME	In <i>techlib.tcl</i> file: Set <i>value</i> to either the full path or relative (./) path (to the current directory) to the net names file for via insertion or to the name of the net names file in your working directory. If specified with NETS_NAME, then NETS_FILE_NAME is used as the path to the net names file.
POWER_VIA_NETS_NAME	Environment variable: Specify in the command line where <i>value</i> is either the full path or relative (./) path (to the current directory) to the file that contains the nets for via insertion or to the name of the net names file in your working directory. This variable performs the same functionality as POWER_VIA_NETS_FILE_NAME.
NETS_NAME	In <i>techlib.tcl</i> file: Set <i>value</i> to either the full path or relative (./) path (to the current directory) to the net names file for via insertion or to the name of the net names file in your working directory. If specified with NETS_FILE_NAME, then NETS_FILE_NAME is used as the path to the nets name file.
NETS	In <i>techlib.tcl</i> file: Set <i>value</i> to net name(s) for via insertion enclosed in braces ({}) and space separated.
POWER_VIA_BEOL_STACK	Environment variable: Specify in the command line where <i>value</i> is the BEOL stack name for your design.
BEOL_STACK	In <i>techlib.tcl</i> file: Set <i>value</i> to the BEOL stack name for your design.
LAYER_MAP_FILE	(Required.) In <i>techlib.tcl</i> file: Set <i>value</i> to your <i>layermap.svrf</i> file enclosed in quotation marks (""). Both full and relative (./) paths (to the current directory) are supported.

Table 9-7. Variables for powervia Utility (cont.)

Variable Name	Description
LAYOUT_PRIMARY	In <i>techlib.tcl</i> file: Set <i>value</i> enclosed in quotation marks (" ") to specify a topcell name for the utility to run on. If not specified, the value defaults to "*", the topcell of the input design. See " Layout Primary " in the <i>Standard Verification Rule Format (SVRF) Manual</i> for more information on name conventions.
RDB_FILE	In <i>techlib.tcl</i> file: Set <i>value</i> to the name of the output results database (RDB) file. Starting and trailing whitespaces are ignored in this file. By default, the file name is set to <i>powervia.rdb</i> .
RESOLUTION	In <i>techlib.tcl</i> file: Set <i>value</i> to a positive integer that specifies the step size of the layout grid in database units. If specified, it appears once in the SVRF rule file. Specifying the step size causes the grid step to be square (equal in x- and y-directions).
SVRF_VERSION	In <i>techlib.tcl</i> file: Set <i>value</i> to the Calibre version specified as v $DDDD.D_D^*.D^*$, where "v" is a literal, "D" is a decimal digit, and D^* is one or more decimal digits. If specified, the SVRF VERSION statement with the Calibre version are written to the SVRF rule file. If a Calibre version earlier than specified is used, the tool exits with an error message. If this variable is not specified, no Calibre version checking is done.

Table 9-7. Variables for powervia Utility (cont.)

Variable Name	Description
TEXT_LAYERS	(Required.) In <i>techlib.tcl</i> file: Set <i>value</i> to the <i>text_layers</i> file path. See your Siemens EDA representative to determine which one of these path specifications is used for your foundry flow: <ul style="list-style-type: none"> Specify the full or relative (./) path (to the current directory) to the existing <i>text_layers</i> file. Specify the path including the directory and filename for the <i>text_layers</i> file which needs to be generated.
FILTER_RECT	(Required.) In <i>techlib.tcl</i> file: Set <i>value</i> for via insertion behavior on overlapping shapes with 45 degree (skew) angles: <ul style="list-style-type: none"> “Yes” — Only the rectangular region of the overlapping shapes is considered for via insertion. “No” — The entire region of the overlapping shapes is considered for via insertion.
VIA0i_blockage_layers	In <i>techlib.tcl</i> file: Set <i>value</i> to a string that contains a derivation of layers to be printed as blockage layers in the SVRF file. For example: <pre>set VIA0i_blockage_layers "layer_derivations"</pre> If this variable is not specified, the flow generates the blockage layers by default. The use of this variable is specific to the process flow. Contact your Siemens EDA representative for the variables specific to your process flow.
VIA_VARIABLES	(Required.) In <i>techlib.tcl</i> file: Set <i>value</i> to the path of the input via variables file. See your Siemens EDA representative to determine which input via variables file is used for your flow. Both full and relative (./) paths (to the current directory) are supported.

Table 9-7. Variables for powervia Utility (cont.)

Variable Name	Description
INITIAL_PLACEMENT	In <i>techlib.tcl</i> file: Set <i>value</i> to specify the starting origin of the placement grid with the corresponding via insertion region to one of the mutually-exclusive options: <ul style="list-style-type: none"> • “LL” — Lower left. • “LR” — Lower right. • “UL” — Upper left. • “UR” — Upper right. • “CENTER” — Center. (This is the default origin if the INITIAL_PLACEMENT variable is not specified.)
DISABLE_SPECIAL_FILTERS	In <i>techlib.tcl</i> file: Set <i>value</i> to “Yes” to disable special empty-area via filters. The default is “No”.
DEBUG_DIR	In <i>techlib.tcl</i> file: Set <i>value</i> to control the content and location of the generated intermediate files (<i>via_addition.sum</i> , <i>net_aware_PWR_GND.oas</i> , <i>calibre.log</i>): <ul style="list-style-type: none"> • “NONE” — Specifies to delete all the intermediate files. • <debug_dir> — Creates a directory named <debug_dir>.time_stamp, which contains all the intermediate files. If the DEBUG_DIR variable is not specified, then the flow generates all the intermediate files in your current working directory.
DEBUG_SPACE	In <i>techlib.tcl</i> file: Set <i>value</i> to specify a list of fill vias for which debugging information is written to an output file. This produces an OASIS file with fillable regions and spacing layers that represent empty areas for the specified fill vias (LRG, BAR, SQR...). For example: set DEBUG_SPACE “VIA0i VIA1i” The output filename is specified by the DEBUG_SPACE_FILE variable.

Table 9-7. Variables for powervia Utility (cont.)

Variable Name	Description
DEBUG_SPACE_FILE	<p>In <i>techlib.tcl</i> file:</p> <p>Set <i>value</i> to specify the output filename for DEBUG_SPACE. The file must be OASIS format or an error is generated. By default, the output filename is <i>debug_space.oas</i>.</p> <p>If the <i>techlib.tcl</i> DEBUG_DIR variable is not specified, the output file is generated in the current directory; otherwise, it outputs to the specified debug directory.</p>
VIA_TYPE	<p>In <i>techlib.tcl</i> file:</p> <p>Set <i>value</i> to define a list of via types (LRG, BAR, and SQUARE) to insert in a single run. The sequence of the via types in the list is not important. If this variable is not defined in the <i>techlib.tcl</i> file, the flow inserts square vias by default.</p> <p>The VIA_TYPE variable can be used in combination with the <i>techlib.tcl</i> SELECT_LAYER variable. If a via is defined in SELECT_LAYER and is not defined with a via type in VIA_TYPE, by default the tool inserts square vias for that via. For example:</p> <pre>set SELECT_LAYER "V3 V4 V5" set VIA_TYPE "V4BAR V5LRG V5"</pre> <p>The tool inserts V3 square (default), V4BAR, V5 square, and V5 LRG vias.</p> <p>See “Insertion of Different Via Types (LRG, BAR, SQUARE)” on page 183 for more usage information.</p>

Chapter 10

DFM Terms and Concepts

Certain DFM-specific terms and concepts are defined and described.

Binning	240
Calibre Results	241
Calibre nmDRC Results Summary	241
Calibre Transcript	243
Capture Window	244
Clustering	245
Concurrency for DFM Operations and Statements	245
Critical Area	247
Defect Density	250
Derived Error Layers	251
DFM Annotations	252
DFM Databases	253
DFM Expressions	254
DFM Function Statement and User-Defined Functions	255
DFM Properties	256
Math Operators	257
Partitioning	259
Pattern Substitution in DFM Commands	264
Quality Assessment Metrics	265
Results Databases	267
Runsets	269
Vectors	270
Yield Assessment Metrics	270
YieldServer	271

Binning

Binning refers to organizing statistically similar results into classes.

In DFM, binning is used in the following ways:

- Assessing the impact of random particles on manufacturability.
- Evaluating the level of compliance with recommended rules.
- Ranking errors.

Each bin, or class, is meaningful when at least two types of information are available:

- Definition of which elements belong in the bin and class.
- Statistics on the rate of occurrence of elements in each bin and class.

Binning Random Particles **240**

Binning Recommended Rule Violations **241**

Binning Random Particles

Binning random particles refers to defining ranges of particle sizes that are likely to be encountered during the manufacturing process, and specifying for each bin the probability that defects in that size range are encountered.

Typically, the probability is calculated separately for each layer.

You use binning of random particle sizes as follows:

1. Organize particles into bins based on sizes. (Supplied through the defect radius list in [DFM Critical Area](#).)
2. Calculate, based on empirical data from the fab, the probability that a particle from a specific bin will land on your chip.
3. Use [DFM Critical Area](#) to calculate the portion of the chip that is susceptible to particles of that size. That is, a particle of that size hitting in the susceptible area will cause an open or a short.
4. Use [DFM Analyze](#) to calculate statistics based on probability and critical area to report information such as:
 - Expected yield for the chip.
 - How a specific size (bin) of particle impacts yield.
 - Which areas of the chip are most likely to be involved in defects caused by random particles.

Binning Recommended Rule Violations

The procedure for binning recommended rule violations includes assigning cost factors to ranges of rule conditions, then using those costs as a weight for calculating yield or quality metrics.

Typically, the cost reflects the susceptibility to process variation, the level of OPC required, or the likelihood that a violation of the rule will contribute to chip failure. Refer to “[Writing Recommended Rule Checks](#)” on page 52 and “[Writing Expressions for Scores and Metrics](#)” on page 58 for instructions on writing recommended rule checks and expressions.

You can use binning of recommended rule violations as follows:

Procedure

1. For each rule type, define the general ranges of values that are of interest to you.
2. For each range of values, assign a cost.
3. Use [DFM Measure](#) or other SVRF operations to isolate the areas of the chip that violate each rule.
4. Use [DFM Analyze](#) to calculate statistics based on cost, number, and size of the violation, and to report information such as:
 - Level of compliance with recommended rules.
 - Design modifications most likely to improve chip manufacturability.
 - Most common violation.
 - Cells containing the most violations.

Calibre Results

The Calibre nmDRC hierarchical engine saves results and status information from each run.

The results are reported in the following forms:

- [Calibre nmDRC Results Summary](#)
- [Calibre Transcript](#)
- [Results Databases](#)

Calibre nmDRC Results Summary

The Calibre nmDRC results summary is a text file that the Calibre nmDRC hierarchical engine generates during a run.

This file contains heading information describing the particulars of the run, a runtime warnings list, statistics for the layers before and after processing, and a runtime summary.

You control the name and location of this file through the [DRC Summary Report](#) statement in your rule file.

Calibre Transcript

The Calibre transcript is a text transcript that the Calibre nmDRC hierarchical engine automatically generates at run-time and sends to stdout, which is typically the shell window from which you invoked the application.

During the Calibre run, the transcript displays event logs, warning messages, and summary information. This transcript documents the tasks the application performs: compiling the rule file, loading in the design data, and operating on that data. When you invoke any of the model-based batch tools, it also records the full setup file referenced and the optical and resist models used for simulation. It is good practice to save the transcript by redirecting the output to a file, which you can do when you invoke the Calibre nmDRC hierarchical engine. Methods you can use include:

- Using the pipe and tee syntax:

```
calibre -drc -hier _litho rulefile.drc | tee calibre.log
```

- Using redirection:

csh or tcsh:

```
calibre -drc -hier rulefile.drc >& log
```

ksh or bsh:

```
calibre -drc -hier rulefile.drc > log 2>&1
```

For a more complete description of the Calibre transcript, refer to “[Calibre nmDRC Results](#)” in the *Calibre Verification User’s Manual*.

Notification Messages for DDKs..... 243

Notification Messages for DDKs

When running a Critical Area Analysis (CAA) session, notification messages for DDKs may be written to the Calibre transcript.

You can control the behavior of the messages with the MGC_CAA_DDK_MESSAGES_TREATMENT environment variable. To do this, issue the following at a shell command prompt before starting Calibre Interactive DFM or submitting a batch run:

- In csh or tcsh:

```
% setenv MGC_CAA_DDK_MESSAGES_TREATMENT "string"
```

- In ksh or bsh:

```
% export MGC_CAA_DDK_MESSAGES_TREATMENT="string"
```

where “*string*” can be one of the following (include double quotes):

- “ignore” — Messages do not appear in the transcript. CAA runs to completion.
- “error” — Warning and error messages do appear in the transcript. CAA does not run to completion.
- “warning” — Messages do appear in the transcript. CAA runs to completion.
- “user-defined word” — Messages do appear in the transcript. CAA runs to completion.
- “not specified” — Messages do appear in the transcript. CAA runs to completion. This is the default if a value is not specified for MGC_CAA_DDK_MESSAGES_TREATMENT or if the supplied string is not recognized.

Capture Window

The capture window is a portion of a layout database evaluated or reported on by one of the DFM operations.

You define capture windows through Partitioning—refer to “[Partitioning](#)” on page 259.

Clustering

Clustering refers to organizing geometric data from multiple layers according to proximity. Clustering is used by the DFM RDB and by DFM Property operations. With DFM Property you can also cluster objects based on node number rather than spatial relationship.

- [DFM RDB](#) — Uses clustering to organize data in a [Multilayer RDB](#). By default, the RDB contains one rule check for every cell in the layout with cell data organized into clusters. When ALL CELLS is specified, the RDB is organized into a single rule check with results grouped by cell and cluster. See the command reference for additional keywords that affect clustering.
- [DFM Property](#) — Uses clustering to define the data used for calculating property values.

Each cluster contains exactly one object from the first input layer to the operation, called the primary layer or the driving layer. Additional input layers to the operation are called secondary layers or driven layers. A cluster may contain any number of objects from the additional input layers, or none at all.

Concurrency for DFM Operations and Statements

Concurrency refers to running multiple layer operations simultaneously, thereby reducing processing time.

Whenever Calibre nmDRC performs layer operations, the tool locates all required layer operations within the set that can run concurrently and executes them as a single group. You can optimize your rule file by taking advantage of this feature. Table 10-1 lists the supported concurrency configurations.

Table 10-1. Supported Concurrency for DFM Operations and Statements

Operations	Operations Run Concurrently when they have...
DFM Analyze	Same input layers (in the same order), and same WINDOW, STEP, and INSIDE OF parameters
DFM Measure	N/A
DFM Critical Area	Same input layers, same particle size list, same particle density list
DFM Transition	Same input layer and file parameters
DFM Function	N/A
DFM Grow	N/A

Table 10-1. Supported Concurrency for DFM Operations and Statements

Operations	Operations Run Concurrently when they have...
DFM Expand Enclosure	N/A
DFM Property	Same input layers and same INSIDE OF parameters
DFM RDB	N/A

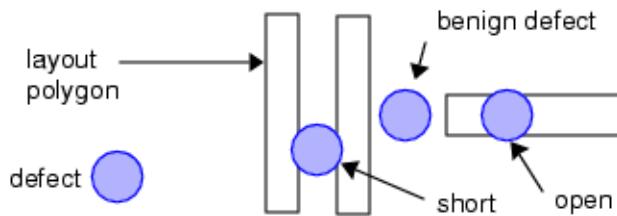
For more information on this topic, refer to “[Concurrency](#)” in the *Calibre Verification User’s Manual*.

Critical Area

Critical Area is the portion of a layout that comprises the locus of defect center points that can cause opens or shorts in the circuit.

Manufacturing defects result from material that is deposited or removed during the manufacturing process. Often, they are random, contaminant particles, but they can also be the result of non-random effects of the fabrication process such as polishing. When defects land in such a way as to cause a short or an open in the circuit, they can be responsible for chip failures.

Figure 10-1. Manufacturing Defects



Exactly which portions of a layout are critical area depends on a number of factors:

- Radius of the defect. The larger the defect is, the larger the critical area becomes.
- Shape of the defect.
- Orientation of the defect.
- The depth of overlap between the defect and an interconnect polygon.
- Failure type (open or short).
- Shapes of polygons on the interconnect layer (wires, general interconnect polygons, vias, and so forth). The wider the separation of the polygons, the smaller the SHORT critical area becomes.

Critical Area for Shorts..... **247**

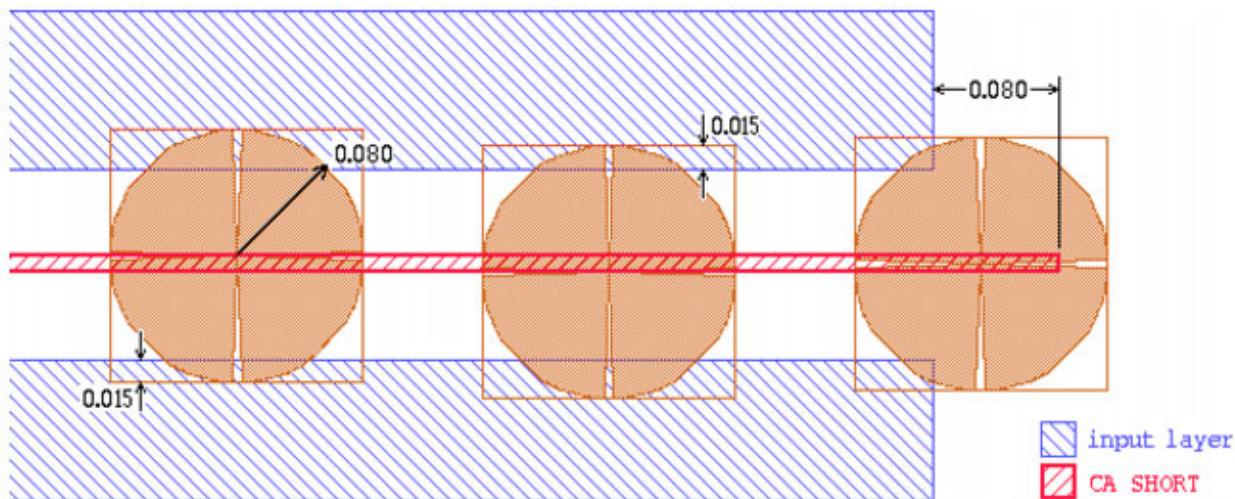
Critical Area for Opens..... **249**

Critical Area for Shorts

The critical area for shorts is the portion of the design where a short will occur if the center point of a contaminant lands there.

Figure 10-2 shows the critical area for a defect shorting parallel wires. Note that the critical area represents only the locus of the defect center points, not the entire defect.

Figure 10-2. Short Circuit Critical Area



By default, DFM Critical Area calculates critical area based on square defect particles. Octagons, which more closely model a circular defect particle, may also be used.

Figure 10-3. Short Circuits Between Parallel Wires

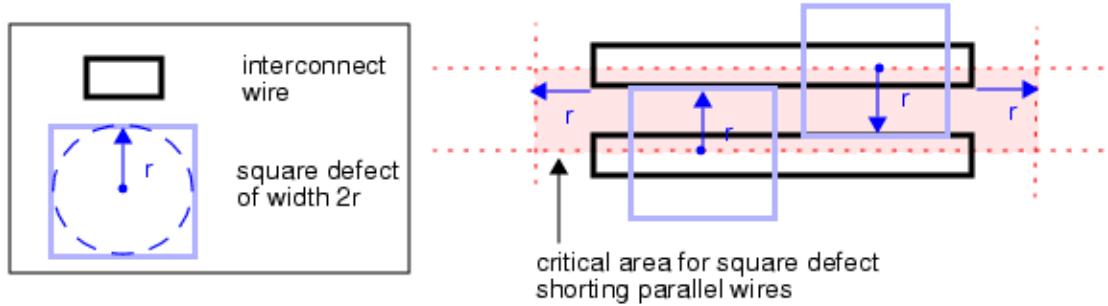
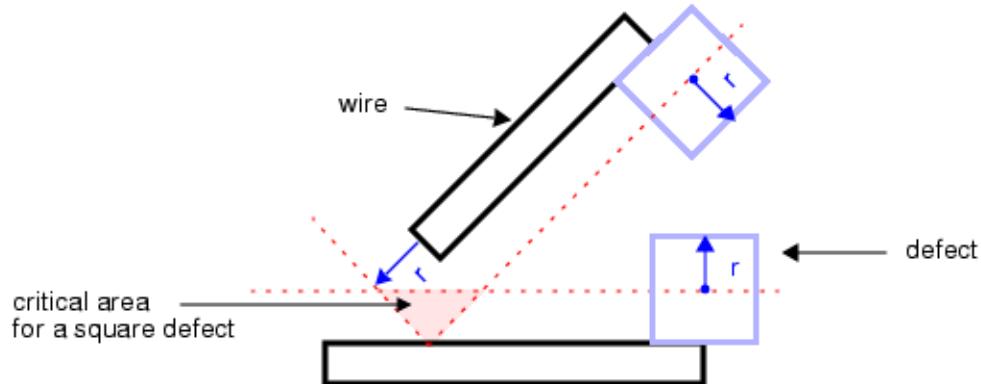


Figure 10-4. Short Circuits Between Non-Parallel Wires



DFM Critical Area addresses one basic type of short: that which occurs between two electrical nets between wires on the same layer.

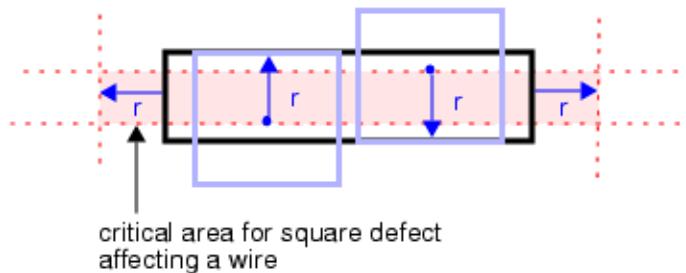
Critical Area for Opens

The critical area for opens is the portion of the design where current will be disrupted if the center point of a contaminant lands there.

Note that the critical area represents only the locus of the defect center points, not the entire defect.

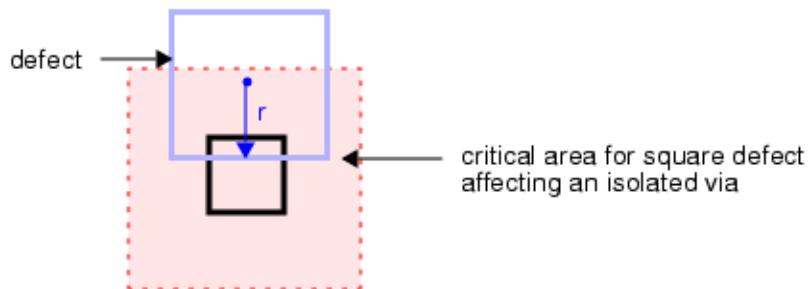
Figure 10-5 illustrates critical area regions for open circuits on a wire.

Figure 10-5. Open Wire Critical Area



The shape of the defect has an effect on the size of the critical area. Using a square defect of width $2r$ results in a slightly larger critical area than a circular defect of radius r , so a square is a more conservative assumption about the defect shape. DFM Critical Area, by default, calculates critical area based on square defect particles. Octagons, which more closely model a circular defect particle, may also be used.

Figure 10-6. Open Via Critical Area



DFM Critical Area addresses three types of open circuit defects:

- Open circuits in wires (defect breaking a wire on a single layer).
- Open circuits in more general interconnect layer polygons.
- Open circuits in vias.

Defect Density

The defect density is the probability that a random particle of a specific size will be present for a given manufacturing process.

Defect densities are used when assessing the impact of random particle defects on yield. They are typically expressed in terms of unit per area, for example:

$$\frac{\text{particles}}{\text{microns}^2}$$

Critical area analysis requires that you know the defect densities for those ranges of particle sizes that are likely to be encountered during the manufacturing process. Taken together, the particle sizes and their associated densities is referred to as *defect density data*.

The easiest way to manage defect density data is through a defect density data file. In most cases, your foundry or process engineer provides you with a defect density file. Every foundry or IDM has their own defect data format, and many of them are encrypted prior to distribution.

Note

 The Calibre Critical Area Analysis tool accepts most encrypted data file formats. Critical area rule decks created using encrypted defect data are also encrypted, ensuring that defect data cannot be extracted from the generated code.

If you do not have foundry defect density data available, you can use the “[CAA NDD Flow](#)” on page 117 to run Calibre CAA as a DFM scoring tool using the standard CAA flow without the requirement of foundry defect density data.

Derived Error Layers

Derived error layers are layers containing clusters of one, two, three, or four edges from either one or two layers.

Derived error layers contain the output of error-directed dimensional check operations. Operations that can generate this type of data include Enclosure, External, Internal, and DFM Measure.

Derived error layers are usually sent directly to the Calibre nmDRC Results Database and are generally not available for other layer operations to manipulate. However, three DFM operations accept this type of data as input:

- [DFM Analyze](#) — Calculates statistics about error data using the COUNT, EC, or EW functions.
- [DFM RDB](#) — Writes error layer data to an RDB.
- [DFM Property](#) — Filters, annotates, or performs both for derived error layers.

Example Derived Error Layer 251

Example Derived Error Layer

Here is an example of a derived error layer.

```
x = internal metal < .10
/* note absence of ( ), [ ], and REGION keywords used for derived layer
output in a dimensional check operation */
```

The layer x is allowed as input to a DFM Analyze operation only if x is used as an argument to a COUNT function in an expression.

For more information on this topic, refer to “[Derived Error Layers](#)” in the *Calibre Verification User’s Manual*.

DFM Annotations

Annotations are data in the form of name and value pairs that you associate with data in a DFM database.

Annotations are associated with either individual layers or with the database as a whole.

You add annotations to a DFM database using the ANNOTATE keyword for the SVRF statements listed in [Table 10-2](#).

Table 10-2. DFM Operations that Support Annotations

SVRF Statement	What is Annotated
DFM Analyze	Layers containing the actual check results (the <i>detail</i> layer).
DFM RDB	Layers containing the geometries on the input layers.
DFM Database	DFM database.

ANNOTATE is a special keyword that is evaluated only when you run Calibre with the -dfm switch. It is supported by a limited set of SVRF commands. Each annotation specification defines one name and value pair. The syntax of the ANNOTATE option is the same for all DFM commands that support them:

`ANNOTATE ['name = "value"]'`

where:

- The brackets ([]) are required.
- The name can be a quoted or unquoted string.
- The value must be a quoted string.

[How Annotations Compare to Other Types of MetaData](#) [252](#)

How Annotations Compare to Other Types of MetaData

Both RDB and DFM databases allow users to attach metadata to some types of objects.

Metadata can be stored as DFM Annotations, DFM properties, comments, or check names. These types of metadata differ from each other in the following ways:

- **Types of Data** — DFM Properties can be strings, vectors, numbers (scalar values) or layer names. Other types of metadata are limited to string values only.

- **Types of Objects** — DFM Properties are attached to individual geometric objects. Check names and comments can only be attached to layers. Annotations can be attached to layers or to an entire database.
- **Database Support** — RDBs support properties, check names, and comments. DFM databases support all these plus DFM Annotations.
- **Multiple Instances** — Use the following guidance:
 - A layer can have only one check name attached to it.
 - A layer can have only one comment attached to it.
 - A layer or database can have an unlimited number of annotations, each stored as a name and value pair. The annotation names need not be unique so it is possible for a layer to have multiple attributes with the same name but different values.
 - A geometric object can have an unlimited number of properties, each stored as a name and value pair. However, each property name must be unique.

Table 10-3. Summary of MetaData Stored In Databases

	Databases	Objects	Data	Stored as...
DFM Annotations	DFM Database RDB	Layers Database	Strings	Name and value pair Allows multiple values for a single name
DFM Properties	DFM Database RDB	Geometric objects	Strings Vectors Numbers Layer Names	Name and value pair Only one value allowed for each name
Check names	DFM Database RDB	Layers / Checks	Strings	Value
Comments	DFM Database RDB	Layers / Checks	Strings	Value

DFM Databases

A DFM database is created by the DFM Database specification statement. It is generated in a calibre -dfm run. The format is proprietary.

The [DFM Database](#) manages the following three types of data:

- Hierarchical layout objects.
- Layout object attributes.
- Layout object properties.

When running `calibre -dfm`, all rule check data is automatically written to the DFM Database by layer.

The DFM Database is different from the [DRC Results Database](#). The former is only generated when `calibre -dfm` is run. The latter is only generated when `calibre -drc` is run. A rule file may support both run modes.

Calibre YieldServer can be used to post-process DFM databases. See “Working with DFM Databases” in the [Calibre YieldServer Reference Manual](#) for details.

DFM Expressions

DFM Expressions are expressions used to define the DFM-related properties, scores, and statistics that describe and report on layout data. They are used in DFM Analyze, the DFM Property family of operations, DFM Spec Fill Optimizer, and DFM Function.

DFM Expressions can include math operators, math functions, built-in and user defined functions, conditional expressions, numeric values, and variables. Expression chains, strings, and vectors can be used in the DFM Property operation. See “[DFM Expressions](#)” in the [Standard Verification Rule Format \(SVRF\) Manual](#) for details.

DFM Function Statement and User-Defined Functions

You can create user-defined DFM functions with the DFM Function statement. The user-defined functions can be used within DFM expressions in DFM Property, DFM Analyze, and other DFM Function statements. User-defined functions make it easier to code complicated expressions and reuse them.

See the [DFM Function](#) statement for complete information on creating and using a user-defined function. Refer to the “[DFM Function Reference](#)” in the *Standard Verification Rule Format (SVRF) Manual* for information on built-in DFM functions.

DFM Properties

DFM properties are a form of metadata associated with layout objects such as polygons, edges, and errors. They are typically numerical values used for calculating scores and filtering data. Each property has a *name* and a *value*. Property *names* are case-sensitive.

For detailed information see “[DFM Properties](#)” in the *SVRF Manual*.

DFM properties can be the following types:

- **Numeric** — Contain floating-point numbers.
- **Net number** — Contain the net ID for a geometry.
- **String** — Contain string values which may be either user-defined or come from a GDS property, an OASIS property, or a text object.
- **Vector** — A vector is a two-dimensional array (or matrix) of numbers composed of tuples. Vectors can contain numeric values, strings, or net numbers.

Math Operators

You can use certain math operators in DFM Expressions to create DFM functions.

The following types of math operators are available for use in DFM functions:

- **Binary Operators** — Take two arguments.
- **Unary Operators** — Take one argument.

Binary Operators [257](#)

Unary Operators [258](#)

Binary Operators

Binary operators take two numeric arguments as inputs and produce a numeric output value.

The precedence of operator execution is determined by the type of operator. In this case, the precedence range is listed from 1 to 4, with 1 having the highest order (first operator to be executed) and 4 having the lowest order (last operator to be executed). If mathematical grouping by parentheses exists, this can override the precedence of operator execution.

Operator usage in expressions can be either conditional or unconditional as specified. Conditional expressions have an if-then construct, whereas unconditional expressions do not.

Table 10-4 lists several binary operators that you can use in DFM Expressions.

Table 10-4. Binary Operators

Operator	Definition	Precedence	Usage
<code>^</code>	power function: $x ^ y$ means x raised to the y power Values of x and y for which results are undefined generate an error The <code>^</code> operator has the same precedence as <code>*</code> and <code>/</code>	1	unconditional expressions
<code>*</code>	multiplication	2	unconditional expressions
<code>/</code>	division	2	unconditional expressions
<code>+</code>	addition	3	unconditional expressions
<code>-</code>	subtraction	3	unconditional expressions
<code>&&</code>	AND (both conditions must be true)	4	conditional expressions

Table 10-4. Binary Operators (cont.)

Operator	Definition	Precedence	Usage
	OR (at least one condition must be true)	4	conditional expressions

Unary Operators

Unary operators take one numeric argument.

These operators are of equal precedence and all are of higher precedence than the binary operators. Unary operators can indicate positive or negative numbers and true or false values. Operator usage is supported in both conditional and unconditional expressions.

There are four unary operators that you can use in DFM Expressions. [Table 10-5](#) lists these operators.

Table 10-5. Unary Operators

Operator	Definition
+	indicates a positive number
-	indicates a negative number
!	returns 1 (true) if its argument is 0; returns 0 (false) otherwise
~	returns 1 (true) if its argument is non-positive; returns 0 (false) otherwise

For example, the unary operator expression !AREA(poly) used in DFM Analyze returns the following values:

- **1** — If there is no poly in the data capture window.
- **0** — If there is poly in the data capture window.

Partitioning

Partitioning refers to dividing geometric data into meaningful and manageable chunks.

Partitioning is used for either of two purposes:

- To divide geometric data into meaningful chunks within which to perform statistical measurements or calculations, as with DENSITY or [DFM Property](#).
- To define how results are organized in a database, as with [DFM Transition](#) or [DFM Measure](#).

The most common partitioning schemes are by WINDOW and by CELL.

The following table shows the default partitioning for each operation that makes use of partitioning.

Table 10-6. Default Partitioning

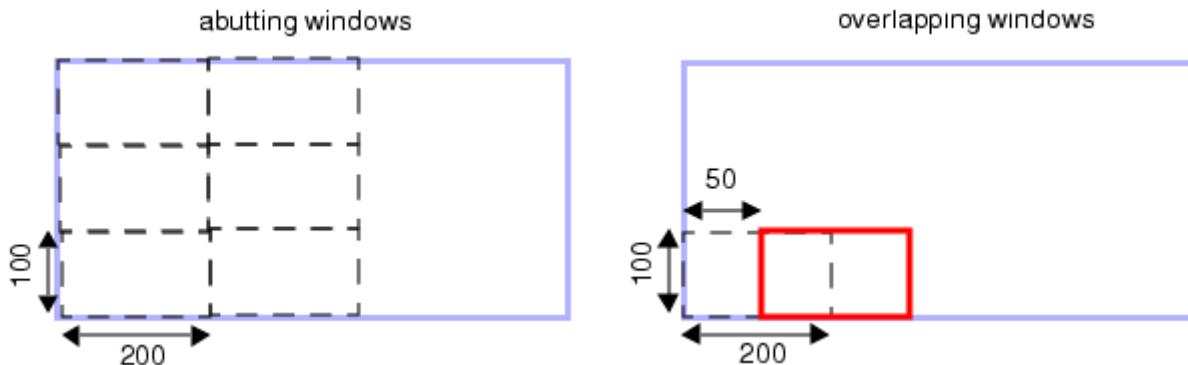
Operation	Default Partitioning
DFM Analyze	Window
DFM Measure	Window
DFM Transition	Window
DFM Property	Polygon Clusters
DFM RDB — Single-Layer RDB Mode	Cell (WINDOW not supported)
DFM RDB — Multilayer RDB Mode	Cell (WINDOW not supported) with cell data organized into clusters of Geometries or errors
DFM RDB — Transitions RDB Mode	Window

Partitioning by Window	260
Partitioning by Cell	263

Partitioning by Window

Partitioning by WINDOW refers to breaking design data into rectangular areas, or capture windows, of a specified size.

Most often data is partitioned into either abutting windows or overlapping windows, however it is possible to partition a design into non-contiguous windows if needed.



Partitioning by window works as follows:

- You specify the portion of the data that is of interest.
- You define either the dimensions of a rectangular window, or the actual number of windows¹ to be fit into the area to be evaluated.
- The tool creates one window oriented at the lower left corner of the area of interest. It then creates additional windows by moving across the data by incremental steps that you specify.
- At each step the tool collects statistics about the data in window or writes results pertinent to that data to the RDB.

WINDOW is the default partitioning for operations that support the WINDOW keyword such as Density, DFM Analyze, DFM Measure, and DFM Transition. However, the default window size is the extent of the input data, which means that by default, data is partitioned into a single window.

Partitioning by window requires three specifications:

- A boundary in which to move the window.
- A window size,
- A step size by which to iteratively move the window through the data.

WINDOW Size..... 261

1. Defining the number of windows instead of the actual window dimensions is only supported for the DFM Analyze operation.

STEP Definition	261
Number of Windows	262
Design Data Boundaries	262

WINDOW Size

You can specify the capture window size through the arguments to the WINDOW keyword.

There are four possibilities:

- **WINDOW x y** — The window is a rectangle that is x user units long and y user units high.
- **WINDOW w** — The window is a square with edges w user units long.
- **WINDOW** — The window is the entire area being evaluated.
- **Not Specified** — The window is the entire area being evaluated.

If a WINDOW parameter exceeds the boundary size in the x-direction, that is, if the WINDOW parameter $x > (\text{BX2} - \text{BX1})$, then both x parameters for WINDOW and STEP are set to $\text{BX2} - \text{BX1}$. Parameters are treated similarly for the y-direction.

STEP Definition

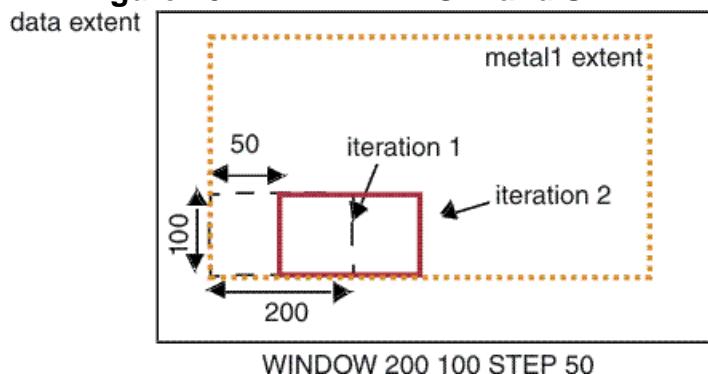
You control the step size by which to iteratively move the capture window through the data using the arguments to the STEP keyword, which you can only use with WINDOW.

There are three possibilities:

- **STEP x y** — The window moves in separate x and y increments, in user units.
- **STEP w** — The window moves in equal increments horizontally and vertically, where w is in user units.
- **Not Specified** — The window moves in increments that match the window dimensions. This results in windows that abut in both directions.

Note that if both WINDOW and STEP are specified, then the STEP parameters must evenly divide the corresponding WINDOW parameters.

Figure 10-7. RDB WINDOW and STEP



If moving the window by the step results in a window that extends beyond the boundary of the data of interest, the operation truncates the window to ensure it contains only data inside by the boundary.

Number of Windows

For the DFM Analyze operations, you have the option to control the capture window size by defining the number of windows into which the area under investigation is to be divided.

The operation uses this information to calculate the required window size.

There are three possibilities:

- NUMWIN *n* — The area under investigation is divided into *n* by *n* windows.
- NUMWIN *nx ny*— The area under investigation is divided into *nx* windows in the x direction and *ny* windows in the y direction.
- NUMWIN — The window is the entire area being evaluated. This is equivalent to specifying WINDOW with no arguments.

When you specify NUMWIN for DFM Analyze, you cannot use STEP. Instead, you can specify the OVERLAP. This is the fractional amount by which a data capture window overlaps the previous data-capture window. As with NUMWIN, you can specify a single value to be used as the overlap in both directions, or specify a separate value for the overlap in the x and y directions. These values must be ≥ 0 and < 1 .

Design Data Boundaries

You specify the boundary of the design data to be evaluated using the INSIDE OF keyword, which you can only use with WINDOW.

There are four possibilities:

- INSIDE OF $x1\ y1\ x2\ y2$ — The area of interest is bounded by a valid orthogonal rectangle boundary expressed as a pair of x , y coordinates. The values must be expressed in user units, with negative values enclosed in parentheses.
- INSIDE OF EXTENT — The area of interest is bounded by the total rectangular extent of the input layers for the operation—see “[Extent](#)” in the *Standard Verification Rule Format (SVRF) Manual*.
- INSIDE OF LAYER $layerB$ — The area of interest is defined through polygons on the specified layer. The parameter $layerB$ must be an original or derived polygon layer and may not be an input layer. The operation computes the individual extents of each polygon on the specified layer and concatenates them. The complete set of concatenated extents is used as the boundaries for the capture window. If the extents of the polygons are disjoint, the capture window steps through each extent region separately.
- Not Specified — The boundary is the database extent read in at runtime. Note that only layers that appear in rule checks selected to participate in the run, or which are required for connectivity, are used to compute the extent of data read in. In other words, Calibre only reads in what it needs from your layout database, and it uses that to compute the layout data extent.

Partitioning by Cell

Partitioning by cell refers to chunking design data on a per-cell basis.

The cell method is triggered with the BY CELL keyword.

- When used for organizing data in a results database, causes results to be partitioned on a cell-by-cell basis.
- When used with DFM Analyze, uses the rectangular extent of each cell as a separate capture window, thereby creating one rectangular capture window per cell.

Internal heuristics determine where the cell is in the hierarchy that Calibre creates during the run. Each cell, whether it is a cell in the original database, or a pseudo-cell that Calibre creates, is used as a data capture window, and the DFM operation results are associated with the cell in which they occur.

Results vary depending on how you choose to handle layout database hierarchy. You must consider both the original database hierarchy and the internal hierarchy of pseudo cells created by the Calibre hierarchical processing algorithms. The following table lists the options available to you.

Table 10-7. Options for Partitioning BY CELL

Option	Keyword	Description	Comment
Partition only based on the original database hierarchy.	NOPSEUDO	Flattens pseudo-cells up to the next level of hierarchy.	These pseudo-cells appear in the RDB by default.
Calculate property values based only on layer data unique to that cell.	NOHIER	Subhierarchy within each cell is not included in Measurement Functions calculations.	DFM Analyze only.
Calculate property values based on all data within the cell.	ALLDEPTH	Measurement Functions return the sum of all values from the cell and its subhierarchy.	DFM Analyze only.

DFM operations with the BY CELL keyword are processed internally as INSIDE OF EXTENT, and the NOPSEUDO keyword is ignored if your run is flat.

Continuing with the previous example, here is how you could specify a DFM operation with a cell-based RDB partition:

```
my_rule {
    DFM MEASURE M1
    ...
    RDB my_rdb_results MAXIMUM 100 NOEMPTY /* output 100 results maximum
    per-cell, but not empty partitions */
    BY CELL           // partition results by cell
}
```

Pattern Substitution in DFM Commands

Pattern substitution refers to a string processing feature that replaces a predefined string with a predictable value.

Pattern substitution is generally used within a larger string to create automatically generated names.

Several DFM SVRF statements support string substitution within RDB filenames. The pattern substitution facility replaces all occurrences of the pattern “%_t_” in the filename with the name of the top cell for the design. The substitution is case-insensitive. For example, both “%_t_” and “%_T_” are equally valid patterns.

Pattern substitution is available in the following DFM commands:

- [DFM Analyze](#)
- [DFM Spec Fill](#)

- DFM Measure
- DFM RDB
- DFM Transition

Example:

```
LAYOUT PRIMARY "ram1"
DFM_CHECK {
  DFM RDB POLY "%_t_.poly.all"
  DFM ANALYZE POLY RDB ONLY "%_t_.poly.analyze"
}
```

The DFM statements in this example create the following RDB files: *ram1.poly.all* and *ram1.poly.analyze*.

Quality Assessment Metrics

Quality assessment metrics are metrics that provide a *relative* grading of designs with respect to the factors that you define to be important.

Some of the types of factors they can reflect include:

- Compliance with recommended rules as follows:
 - Violation count
 - Violation density
 - Weighted violation densities:
 - Weighting of violations within rules, by bin, run length, and so on.
 - Weighting of violations of many rules by impact, bias, and so on.
- Presence of critical area as follows:
 - Critical area density (critical area/total area)
 - Weighted critical area density (sensitivity to smaller particles weighted more heavily)
- Level of redundancy provided for the following:
 - Vias
 - Contacts

You define quality assessment metrics in the form of **DFM Expressions** to be evaluated by the **DFM Analyze** operation. Outputs from recommended rules analysis and critical area analysis serve as the inputs to be analyzed.

Quality assessments metrics do not predict actual yield. However, properly designed quality assessment metrics do provide several benefits over simple yield assessments:

- You can design quality assessment metrics around the available information (a complete yield specification from the foundry is not necessary).
- They reflect in-house knowledge, judgment, or intuition rather than rigorous statistical characterization.
- They can predict the following:
 - Given two designs, which is more manufacturable?
 - Which areas of a chip present the greatest manufacturability challenges?
 - Which issues result in the greatest impact?
 - How does yield impact vary by cell?
- They provide designers with the direct feedback they need if they are to develop an awareness of how their design practices impact yield and manufacturability.

Results Databases

A results database is a database containing data that results from processing original or derived layers with one or more Calibre nmDRC or DFM operations.

A results database can be any of the following types of databases.

Table 10-8. Summary of Results Databases

Results Database	Created by...
DRC Results Database	calibre -drc
DRC Check Map	calibre -drc
DFM Database	calibre -dfm
DFM RDB	calibre -drc or calibre -dfm. In the latter case, the data is stored in the DFM Database on a per-layer basis rather than per-check.
Side RDB files	Layer operations having the RDB keyword, such as Density, DFM Analyze, DFM Measure, DFM Transition, DFM Spec Fill, and Net Area Ratio. See the individual command reference. calibre -drc and -dfm observe these keywords.

For details about the structure of DRC results databases, see the “[DRC Results Database](#)” section of the *Calibre Verification User’s Manual*.

Calibre nmDRC Results Database	267
DFM RDB Files	269

Calibre nmDRC Results Database

When running calibre -drc, the default behavior is to write all rule check results to the DRC Results Database file.

DRC results databases are usually one of three formats: ASCII, GDS, or OASIS. You must specify the path and format for this database using the [DRC Results Database](#) statement in your rule file.

- **ASCII** — The results of each rule check are saved under one rulecheck section headed by the name of the rule check. These databases are typically viewed in Calibre RVE; however, they are also viewable in Calibre DESIGNrev.
- **GDS** — By default, all results from all rule checks are merged onto a single layer and written to layer 0 of the GDS file. You can specify a separate layer, a datatype, or both for each rule check using the [DRC Check Map](#) statement.

- **OASIS** — By default, all results from all rule checks are merged onto a single layer and written to layer 0 of the OASIS file. You can specify a separate layer, a datatype, or both for each rule check using the DRC Check Map statement.

Only rule check output is written to a Calibre nmDRC Results Database. You can output any original or derived layer using the Copy or DFM Copy operation. (All layers are stored in a DFM Database by default, so copying layers is unnecessary in a calibre -dfm run.)

If an operation supports the RDB keyword, RDB ONLY specifies that the data only goes to the side RDB, not to the DRC Results Database.

DFM RDB Files

A DFM RDB file is a results database that you create in addition to the DRC Results Database when using DFM layer operations in a calibre -drc run. If used in a calibre -dfm run, DFM RDB output data is stored in a DFM Database on a per-layer rather than per-check basis. Additionally, certain layer operations support an RDB keyword. This keyword generates a results database similar to the DFM RDB operation.

DFM RDB has several syntactical forms that are suited for different purposes. In addition, it supports many secondary keywords for configuring the output. DFM RDB has these special modes of note:

Table 10-9. DFM RDB Modes

Mode	Contains
DFM RDB — Single-Layer RDB	Polygons, edges, or errors
DFM RDB — Multilayer RDB	Clusters of polygons or edges
DFM RDB — Transitions RDB	Transitions

The *SVRF Manual* contains more details.

Some operations available through [Calibre YieldAnalyzer](#) and [Calibre YieldEnhancer](#) can be configured to generate so-called “side RDBs” through the RDB secondary keyword. These are separate results databases containing data from individual layer operations. They are similar to DFM RDB databases in many respects, but they are customized for each layer operation they support.

Table 10-10. Operations Generating Side RDBs

Generated by	Contains
DFM Analyze ... RDB	Scores and statistics in a Scores RBD
DFM Measure ... RDB	Check results in a Check Results RDB
DFM Transition ... RDB	Transitions in a Transitions RDB

Runsets

A runset is a text file created by Calibre Interactive DFM to store the settings you specify in the GUI. It is a record of the data displayed in the user interface at the time the runset was created.

Using a runset provides the following benefits:

- Reduces your work time because you no longer need to repeatedly enter all the information necessary to run Calibre.

- Helps you manage your design environment. You can create a different runset for each design situation (process, layers, and so on).

For complete information, see “[About Calibre Interactive Runsets](#)” in the *Calibre Interactive User’s Manual*.

Vectors

A vector is a two-dimensional array of numbers in which one dimension, the tuple size, is fixed and the other dimension, the number of tuples, is variable.

In other words, each tuple is a row of the two-dimensional array and the vector can have arbitrary number of rows (or tuples), but all tuples in the same vector have the same size.

Within this documentation, the following conventions are followed with respect to vectors:

- { } — Notation used to indicate a vector.
- < > — Notation used to indicate a tuple within a vector. Arguments within a tuple are separated by a single space.
- **Argument** — A single value within a tuple.
- **Tuple** — A row in the two-dimensional array.
- **TupleSize** — The number of arguments in each tuple in the vector.

Example 10-1. Vectors

```
{<1 2 3> <4 5 6>}
```

In this example, the number of tuples is 2 and the tuple size 3.

See “[DFM Function Reference](#)” in the *Standard Verification Rule Format (SVRF) Manual*.

Yield Assessment Metrics

Yield assessment metrics are metrics that predict actual yield for specific designs.

Yield assessment can be expressed through many different ways, including the following:

- Percent Yield (%)
- Percent Failure (%)
- Failure Density (#/A)

You define yield assessment metrics through a yield model equation (see “[DFM Expressions](#)” in the *SVRF Manual*) that is evaluated by the [DFM Analyze](#) operation. The DFM TVF package provides several built-in yield models, including:

- POISSON
- SEEDS
- EXPONENTIAL
- NEGATIVE-BINOMIAL *alpha*

where:

alpha — A user defined value for the probability mass function parameter.

$$0 > \text{alpha} > 1$$

- MURPHY
- BOSE-EINSTEIN
- USER *exp*

where:

exp — A yield model equation written in terms of *lambda*, which Calibre calculates and makes available to you through the variable \$lambda. Remember to use the EXP function when defining the equation.

Calculating yield metrics requires that you have access to sensitive data from the fab or foundry, such as defect density size distribution and recommended rule failure rates.

Yield assessment metrics are *not* a prerequisite for using DFM. In fact, most organizations prefer to avoid yield as a metric because it may be used to infer a guarantee of cost. When data from the fab is not available, you can run DFM using [Quality Assessment Metrics](#) instead.

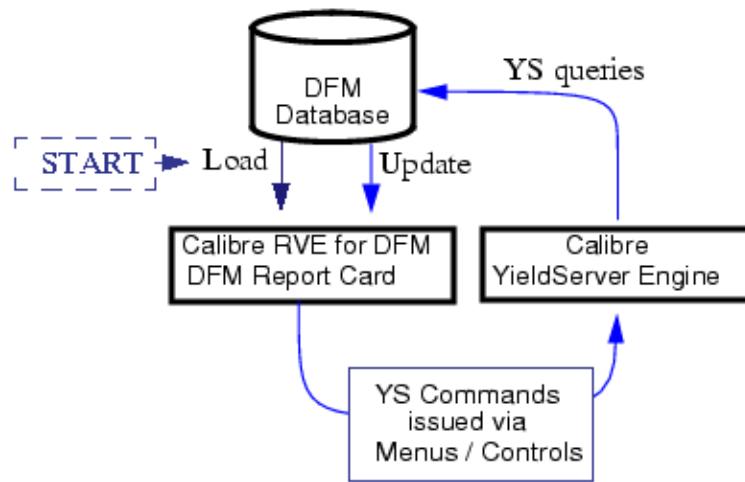
YieldServer

Calibre YieldServer is a software system designed specifically for managing the layout design data required when designing for improved manufacturability and yield.

There are four components to Calibre YieldServer:

- DFM Database
- Calibre YieldServer Engine
- YieldServer Command Language
- DFM Report Card in the Calibre RVE for DFM Graphical User Interface

Figure 10-8. The Calibre YieldServer System



For a complete discussion of what the Calibre YieldServer is and how you can use it, refer to the [*Calibre YieldServer Reference Manual*](#).

Chapter 11

DFM Reference

This section includes DFM reference material, including certain application statements and file format specifications.

For a complete list of the specification statements used in DFM applications, see “[DFM Specification Statements](#)” in the *Standard Verification Rule Format (SVRF) Manual*. Most layer operations that start with “DFM” are listed in “[Auxiliary Layer Operations](#)” in the *SVRF Manual*.

DFM Via Shift Commands	274
DFM Reshape Commands	298
DFM RDB Reference.....	316
DFM File Formats	347
DFM Analysis Reference.....	375

DFM Via Shift Commands

The reference information for the Calibre DFM Via Shift commands is provided. These commands use a Calibre YieldEnhancer license.

For other Calibre YieldEnhancer via applications, see also “[Layout Enhancement](#)” on page 81 and “[Calibre YieldEnhancer PowerVia Flow](#)” on page 176.

Introduction to DFM Via Shift	274
DFM Spec Via Shift	275
DFM Via Shift	280

Introduction to DFM Via Shift

The DFM Via Shift and DFM Spec Via Shift SVRF commands are used together with Calibre YieldEnhancer after RET metal biasing and retargeting to improve via locations and metal coverage while observing mask rule check (MRC) constraints.

These two commands perform the functionality of the DFM Via Shift operation as follows:

- **DFM Via Shift** — Moves vias after biasing and retargeting and optimizes via enclosures by their intended upper and lower metals.
- **DFM Spec Via Shift** — Provides the various MRC spacing constraints required and is used by the DFM Via Shift command.

The DFM Via Shift operation includes the following capabilities:

- Automatic optimization of via placement respective of their upper and lower metal coverage
- Manipulation of square and rectangular vias
- Support of common MRC constraints
- Limitation of hierarchical data promotion where possible
- Implementation of trade-offs to improve coverage of all vias within interactive-via clusters
- Implementation of intelligent area coverage to optimize via placement as opposed to simple metal enclosure rules

DFM Spec Via Shift

SVRF commands: Specification statement

Specifies MRC constraints to improve via location and metal coverage, preventing MRC violations. Referenced by DFM Via Shift. Uses a Calibre YieldEnhancer license.

Usage

Usage for Interaction of Movable Square Vias

DFM SPEC VIA SHIFT

```
spec_name
SQUARE MRC_rules
[CENTER_SPACE center_space_value]
```

Usage for Interaction of Movable Rectangular Vias

DFM SPEC VIA SHIFT

```
spec_name
LONG_LONG MRC_rules
LONG_SHORT MRC_rules
SHORT_LONG MRC_rules
SHORT_SHORT MRC_rules
[CENTER_SPACE center_space_value]
```

Usage for Interaction of Movable Rectangular to Movable Square Vias

DFM SPEC VIA SHIFT

```
spec_name
LONG_SQUARE MRC_rules
SHORT_SQUARE MRC_rules
[CENTER_SPACE center_space_value]
```

Usage for Interaction of Movable Square Vias to Unmovable Vias With Unknown Size

DFM SPEC VIA SHIFT

```
spec_name
SQUARE MRC_rules
LONG_SQUARE MRC_rules
SHORT_SQUARE MRC_rules
[CENTER_SPACE center_space_value]
```

Usage for Interaction of Movable Rectangular Vias to Unmovable Vias With Unknown Size

DFM SPEC VIA SHIFT

```
spec_name
LONG_LONG MRC_rules
LONG_SHORT MRC_rules
```

SHORT_LONG MRC_rules
SHORT_SHORT MRC_rules
LONG_SQUARE MRC_rules
SHORT_SQUARE MRC_rules
[CENTER_SPACE *center_space_value*]

Where **MRC_rules** has the following syntax:

```
{SPACE value |  
    OPPOSITE value [opposite_options] |  
    OPPOSITE value [opposite_options] SPACE value}
```

Arguments

- ***spec_name***

A required argument labeling the specifications for subsequent reference by a DFM Via Shift command. Each DFM Spec Via Shift statement must have a unique name. Multiple DFM Via Shift commands with different input layers may share a single DFM Spec Via Shift statement.

- **SQUARE**

A required keyword for square via interaction triggering a spacing measurement of the distance from one square via to another square via.

- **LONG_LONG**

A required keyword for rectangular via interaction triggering a spacing measurement of the distance from the longer side of one rectangular via to the longer side of another rectangular via.

- **LONG_SHORT**

A required keyword for rectangular via interaction triggering a spacing measurement of the distance from the longer side of one rectangular via on the first layer to the shorter side of another rectangular via on the second layer.

- **SHORT_LONG**

A required keyword for rectangular via interaction triggering a spacing measurement of the distance from the shorter side of one rectangular via on the first layer to the longer side of another rectangular via on the second layer.

- **SHORT_SHORT**

A required keyword for rectangular via interaction triggering a spacing measurement of the distance from the shorter side of one rectangular via to the shorter side of another rectangular via.

- **LONG_SQUARE**

A required keyword for rectangular to square via interaction triggering a spacing measurement of the distance from the longer side of one rectangular via to a square via.

- **SHORT_SQUARE**

A required keyword for rectangular to square via interaction triggering a spacing measurement of the distance from the shorter side of one rectangular via to a square via.

- **MRC_rules**

A required argument set of spacing rules, where MRC_rules is specified by one or both of the following:

SPACE value

A required keyword specifying the Euclidean measurement of the MRC region around the via shape. The SPACE measurement includes non-projecting edges and corners. The value must be a positive number in user units. The SPACE keyword may be specified with or without the OPPOSITE keyword.

OPPOSITE value [EXTEND BY *extend_by_value* | PROjecting *projection_length*]

A required and optional keyword set specifying the perpendicular extension of the MRC region from the edge of a via shape, but not along the edge of a via shape. The value is an extension distance in user units and must be a positive number. The OPPOSITE keyword can be specified with or without the SPACE keyword.

EXTEND BY *extend_by_value*

An optional keyword specifying the extension of the MRC region measured along the edge of a via shape. The value is an extension distance in user units and must be a positive number.

PROjecting *projection_length*

An optional keyword specifying the perpendicular spacing between two vias using the OPPOSITE spacing rule when the edge of one via projects onto the edge of another via. The spacing rule applies when the length of any projection conforms to a valid $\text{projection_length} \geq 0$. A different spacing rule can be specified for each *projection_length*, but the *projection_length* must be non-overlapping. There should be no gaps or overlaps among the different projection lengths. The spacing rule must be complete and should cover all projection scenarios ≥ 0 (projecting and non-projecting). The length is a distance in user units and must be a positive number.

Key points on spacing rules usage:

- When OPPOSITE is used, either EXTEND BY or PROjecting can be specified, but not both.
- When PROjecting is specified, the spacing rule must be complete, meaning that the spacing rule should be specified for both the projecting and non-projecting scenarios:
 - The projecting spacing rule is specified using OPPOSITE with the PROjecting option.
 - The non-projecting spacing rule is specified using OPPOSITE or OPPOSITE with EXTEND BY.

- When PROJecting is *not* specified, the spacing rule using OPPOSITE or OPPOSITE with EXTEND BY applies to both the projecting and non-projecting scenarios.
- When SPACE is specified, the spacing rule applies to both projecting and non-projecting scenarios, irrespective of the PROJecting option.

See “[Example 2 — Projecting and Non-Projecting Spacing Rules](#)” on page 279.

- **CENTER_SPACE *center_space_value***

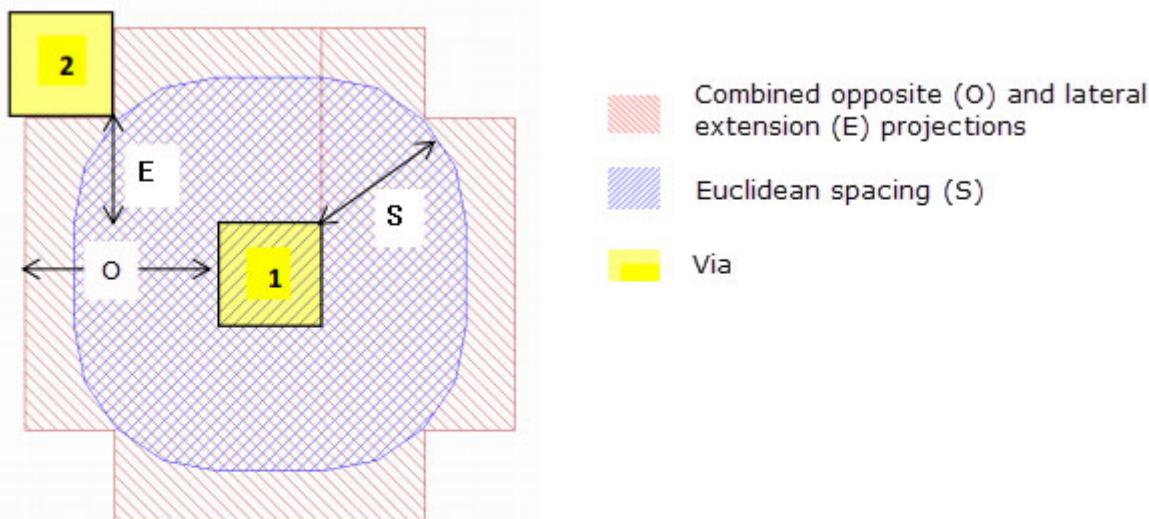
An optional keyword specifying a center-to-center spacing constraint between two vias. The value is in user units and must be a positive number.

Description

A statement specifying spacing values and constraints referenced by subsequent [DFM Via Shift](#) commands to improve via location and metal coverage, preventing MRC violations. Any number of DFM Spec Via Shift statements may be specified.

In [Figure 11-1](#), the MRC constraints are represented by O (perpendicular opposite projection from the edge of one via to the edge of another via), E (lateral extension of the projection along the edge of a via), and S (Euclidean spacing distance around the via).

Figure 11-1. MRC Zones for Vias



The MRC zones for a via are represented by S (euclidean spacing), O (opposite edge projection), and E (lateral extension of projection). Via 2 lies outside the constraints of via 1, resulting in no MRC violations.

Examples

Example 1 — DFM Spec Via Shift Statement

The DFM Spec Via Shift statement specifies constraints for, and is referenced by, the DFM Via Shift command. The spec_name is highlighted in green.

```
DFM SPEC VIA SHIFT spec_1
    SQUARE OPPOSITE 0.08225 EXTEND BY 0.0235 SPACE 0.0725
    shifted_via = DFM VIA SHIFT
        [via(VIA 0.005) m_low(LOWER) m_up(UPPER) SPEC spec_1]
```

Example 2 — Projecting and Non-Projecting Spacing Rules

The projecting and non-projecting spacing rules are applied.

```
DFM SPEC VIA SHIFT spec_1
// Projecting spacing
SQUARE OPPOSITE 0.031 PROJ >= 0.005
SQUARE OPPOSITE 0.029 PROJ >= 0 < 0.005
// Non-projecting spacing
SQUARE OPPOSITE 0.025 EXTEND BY 0.01
// Via shifting
shifted_via = DFM VIA SHIFT
    [via(VIA 0.005) m_low(LOWER) m_up(UPPER) SPEC spec_1]
```

The spacing rule “OPPOSITE 0.031” applies when the length of the edge projection is ≥ 0.0050 . The spacing rule “OPPOSITE 0.029” applies when the length of the edge projection is ≥ 0 and < 0.005 .

The spacing rule “OPPOSITE 0.025 EXTEND BY 0.01” applies when the via edges are non-projecting.

DFM Via Shift

SVRF Commands: Layer operation

Improves via location and metal coverage without introducing MRC violations. Requires DFM Spec Via Shift for MRC constraint specification. Uses a Calibre YieldEnhancer license.

Usage

```
shifted_via_layer = DFM VIA SHIFT
[map_option]
['via_shift_args']...
[GRID value]
[INVALID_VIA_MIN_SPACING invalid_space_value]
[marker_layer('MASK [oversize] | MARKER marker_size')]
[SPEC spec_name {input_via_layer1 input_via_layer2 |
  ('input_via_layer_set') input_via_layer /
  input_via_layer ('input_via_layer_set') |
  ('input_via_layer_set1') ('input_via_layer_set2')}]...
```

Where *via_shift_args* has the following syntax:

```
input_via_layer('VIA [MAX_SHIFTXY] {max_shift_value1[:max_shift_value2]
  [via_layer_options]}')
[ {lower_metal_layer('LOWER') upper_metal_layer('UPPER')} |
  common_metal_layer('COMMON') ]
[exclusion_layer('EXCLUSION [NO_CORNER_CHOP_OVERLAP]')]
[SPEC spec_name]
[ {TARGET_ENClosure enc_value[:enc_value_long]} |
  {TARGET_ENCclosureXY enc_value[:enc_value_y]} ]
```

Arguments

- ***shifted_via_layer***
A required output layer containing the relocated via shapes.
- ***map_option***
An optional argument set that outputs additional layers containing vias with invalid size and shape from the input layer or separators between via geometries. The map_option arguments are specified as one argument per DFM Via Shift instance. The DFM Via Shift operation and required arguments can be specified with only the map_option for input layer analysis (no shifted vias are output).

INVALID_VIA_SIZE — If specified, the output layer contains vias from the input layer that do not match the user-specified size.

INVALID_VIA_SHAPE — If specified, the output layer contains vias from the input layer that are neither square nor rectangular.

CLUSTER_SEPARATORS — If specified, the output layer contains separators (edge pairs of type-3) connecting vias that form a cluster, where a cluster is a group of vias that interact during the shift. This layer can be used to select certain sizes of via clusters. The clusters are determined using input specifications such as MRC constraints and max_shift.

If map_option is not specified, the output layer contains shifted vias.

See “[Example 4 — Layers for Input Layout Analysis](#)” on page 290.

- ‘[’ *via_shift_args* ‘]’

A required argument set, requiring enclosing brackets. The *via_shift_args* set requires an *input_via_layer*, the maximum shift distance, and a via size specification (for movable layers). Optional arguments include via layer options, via enclosing metal layers, an optional exclusion_layer, an optional SPEC keyword referencing the rules for shifting vias, and an optional target for the via enclosure distance. Any number of *via_shift_args* sets may be specified.

input_via_layer‘(‘**VIA** {[MAX_SHIFTXY] *max_shift_value1*[::*max_shift_value2*] [*via_layer_options*]})‘’

A required keyword and options specifying the maximum distance vias on the *input_via_layer* may be shifted to their new location. The *input_via_layer* must be a polygon layer (layer type-1).

The DFM Via Shift command ignores polygons with skew edges or edge constraints not equal to four (!=4), and in these cases, the MRC constraints are not applied.

Vias of different types or sizes must be specified on different *input_via_layers*. The VIA keyword must be specified with a value:

[MAX_SHIFTXY] *max_shift_value1*[::*max_shift_value2*] — A required value and options in user units specifying the maximum distance that the via can move in the horizontal and vertical direction. The maximum shift values are specified as positive numbers or zero. All non-zero values must be an integral multiple of the *database precision*. If *max_shift_value1* is non-zero, the SPEC, LOWER and UPPER, or COMMON keywords must be specified.

If the MAX_SHIFTXY option is specified before the maximum shift values, then the maximum shift is in the x- and y-direction where *max_shift_value1* is the maximum shift in the x-direction and *max_shift_value2* is the maximum shift in the y-direction for both square and rectangular vias.

If the MAX_SHIFTXY option is *not* specified, then *max_shift_value1* is the maximum shift for the via edge. For rectangular vias only, if *max_shift_value2* is specified, then *max_shift_value1* specifies the maximum shift in the direction perpendicular to the short edge of the via, and *max_shift_value2* specifies the maximum shift in the direction perpendicular to the long edge of the via; otherwise, *max_shift_value1* specifies the maximum shift for all sides of square and rectangular vias.

The via layer is unmovable if the maximum shift in both the horizontal and vertical direction is zero. Specify an unmovable via layer if it interacts with a

shifted via layer and you want to prevent violations between the two via layers. Specifying zero movement disables the SPEC, TARGET_ENCLOSURE, LOWER, UPPER, and COMMON keywords. At least one movable input_via_layer must be specified in a DFM Via Shift operation.

via_layer_options

SIZE *size_value*[::*size_value_long*] — A keyword that is required for movable via layers and optional for unmovable via layers. This keyword specifies the size of the via, where *size_value* is the short-edge length and *size_value_long* is the long-edge length of the via. For square vias, only *size_value* is needed. For rectangular vias, both *size_value* and *size_value_long* are required. The values are in user units and are positive numbers that must be an integral multiple of the *database precision*. If specified, all vias on via layer should be of the given size. Vias that do not match the given size are filtered. See also INVALID_VIA_SIZE and INVALID_VIA_MIN_SPACING.

See also “[DFM Spec Via Shift](#)” on page 275 for the usage syntax to specify both square and rectangular vias (mixed via types) with unknown size (**SIZE** *not* specified) in one unmovable layer.

EXTEND BY *ext_value*[::*ext_value_long*] — An optional keyword specifying to resize both ends of a via edge and apply MRC constraints to the resized edge. The via edges are resized only for the purpose of applying the MRC constraints; the size of the output via is unchanged. The values are in user units and can be a positive number, zero, or a negative number. All non-zero values must be an integral multiple of the *database precision*. A positive value specifies the amount to laterally extend both ends of an edge. A negative value specifies the amount to laterally retract both ends of an edge. Via edges are extended or retracted by applying *ext_value* to both ends of the edge. For rectangular vias only, if *ext_value_long* is specified, then the short edge of the via is extended or retracted by applying *ext_value* to both ends of the short edge, and the long edge of the via is extended or retracted by applying *ext_value_long* to both ends of the long edge. See “[Example 7 — Via Edge Resize for MRC Constraints with EXTEND BY](#)” on page 294.

CHOP BY [*chop_value*[::*chop_value_long*] [TYPE {CROSS | OCTAGON}]]

[**SHRINK** *shrink_value*[::*shrink_value_long*]] — An optional keyword specifying to chop (alter) the via shape internally (during the operation) to improve centering inside the common metal of both covered and partial vias. The shape of the output via remains unchanged. This keyword should only be specified for movable via layers. The values must be positive (exception noted), less than half of the edge length, and an integral multiple of *database precision*.

The *chop_value* is the corner-chop distance of the short side from the corner and is applied to both corners of the side. The *chop_value_long* is the corner-chop distance of the long side from the corner and is applied to both corners of the side. For square vias, only *chop_value* is needed. For rectangular vias, both *chop_value* and *chop_value_long* can be specified. If *chop_value_long* is not specified, *chop_value* is applied to the long side of a via. Using the chop values, the via shape is internally considered as either CROSS or OCTAGON by the tool. The TYPE keyword is optional and defaults to CROSS.

The shrink_value is the side-chop distance by which the short side is reduced from both ends as specified with the SHRINK option. The shrink_value_long is the side-chop distance by which the long side is reduced from both ends. For a square via, only shrink_value is needed. For rectangular vias, both shrink_value and shrink_value_long can be specified, where one of these values can be zero. If shrink_value_long is *not* specified, shrink_value is applied to the long side of a via. The side-chop values can be specified with or without a corner-chop value.

The following are examples of CHOP BY:

- Only corner-chop is used:

```
CHOP BY 0.005
```

- Only side-chop (shrink) is used:

```
CHOP BY SHRINK 0.001
```

- Only applies side-chop (shrink) to the short side of a rectangular via:

```
CHOP BY SHRINK 0.001::0.0
```

- Both corner-chop and side-chop (shrink) are used:

```
CHOP BY 0.005 SHRINK 0.001
```

Note

 When specifying both corner-chop and side-chop (shrink), corner-chop must be specified before side-chop.

```
{lower_metal_layer('LOWER') upper_metal_layer('UPPER')} |  
common_metal_layer('COMMON')
```

These arguments are required only for non-zero max_shift values. Specifies either the lower_metal_layer and upper_metal_layer, or a common_metal_layer which is a logical AND of the lower_metal_layer and upper_metal_layer. These layers enclose shapes on the input_via_layer. The lower_metal_layer, upper_metal_layer, and common_metal_layer must be polygon layers (layer type-1).

```
exclusion_layer('EXCLUSION [NO_CORNER_CHOP_OVERLAP]')
```

An optional argument enabled only for via_shift_args sets specifying non-zero max_shift values. Specifies a layer defining regions where no vias are permitted to enter during via shifting. If vias originally overlap an exclusion_layer, they may or may not be shifted out of the exclusion region. The exclusion_layer must be a polygon layer (layer type-1).

The NO_CORNER_CHOP_OVERLAP option can be specified in the case when CHOP BY is used. If specified, the operation uses the corner-chop via shape instead of the actual via shape to determine the overlap of the exclusion region with the via. The side-chop (shrink) via shape from CHOP BY is not considered, and only the CROSS shape type is supported.

```
SPEC spec_name
```

This keyword is required for via_shift_args sets specifying non-zero max_shift values. It should be specified for movable via layers. References a set of MRCs specified in a DFM Spec Via Shift statement that prevent violations between vias defined on input_via_layers.

{TARGET_ENCclosure *enc_value*[::*enc_value_long*] } | {TARGET_ENCclosureXY *enc_value*[::*enc_value_y*] }

An optional keyword choice enabled only for via_shift_args sets specifying non-zero max_shift values. Specifies a target distance for the enclosure of the via edges with respect to the common metal. You can specify either TARGET_ENC or TARGET_ENCXY for a layer type. The tool uses this specification to control the via enclosure but does not guarantee a particular enclosure for the via. The enclosure values are in user units and can be zero or a positive number. All non-zero values must be an integral multiple of the *database precision*.

For TARGET_ENC, if *enc_value_long* is not specified, then *enc_value* is the targeted enclosure for all via edges, else, *enc_value* is the targeted enclosure for short edges of the via, and *enc_value_long* is the targeted enclosure for long edges of the via.

For TARGET_ENCXY, if *enc_value_y* is not specified, then *enc_value* is the targeted enclosure in both the x- and y-direction, else, *enc_value* is the targeted enclosure in the x-direction, and *enc_value_y* is the targeted enclosure in the y-direction. The x- and y-direction is defined from the top cell view.

- **GRID *value***

An optional keyword that specifies the resolution (layout grid step-size) of via shifting movement used during DFM Via Shift. The value is the number of database units per grid step. For example, for a resolution of one database unit (1 dbu), specify a GRID value of 1. Similarly, for a resolution of 2 dbu, specify a GRID value of 2. The shifted via is aligned to the layout grid based on this value. The default resolution is 1 dbu, which means by default, the via can take any position on the layout grid.

- **INVALID_VIA_MIN_SPACING *invalid_space_value***

An optional keyword that specifies the minimum spacing of movable vias with invalid vias. Invalid vias are defined as vias that do not match the user-specified size (SIZE keyword), as well as vias that are neither square nor rectangular. Invalid vias are unmovable and maintain a minimum spacing with movable vias, based on this option. The invalid_space_value is a positive number specified in user units. By default, the minimum spacing for an invalid via with other vias is 1 dbu, which is equal to 1/**PRECISION** in user units.

- ***marker_layer*‘(MASK [*oversize*] | MARKER *marker_size*)’**

An optional argument and keywords that specify a marker region or regions. The *marker_layer* can be specified for either mask or marker regions using the MASK or MARKER keywords.

‘(MASK [*oversize*])’ — If specified, every polygon on the *marker_layer* creates one marker region that is a Manhattan rectangle equal to the extent of the polygon. If the oversize value is specified, the extents are oversized by that value (similar to

specifying the **SIZE** operation) before being used as marker regions. The oversize value, if specified, must be a positive floating-point number in user units.

'('MARKER *marker_size*'') — If specified, the marker regions are squares centered at the center point of each marker polygon. The *marker_size* value defines the size of the squares and must be specified as a positive floating-point number in user units.

As part of a flow, the marker regions are used to preserve the SRAM array consistency, such that all identical units (repeated polygon patterns) in the array get the same via shift. The DFM Via Shift operation takes the marker region as input and generates the shifted via result in the marker region.

The output of the DFM Via Shift operation contains only the vias (or the partial vias, if clipped by a marker boundary) that are inside the marker region after the shift. When a via moves (shifts) into or out of a marker region, the output contains only the part of the via that remains inside the marker region.

The marker flow uses an external operation in the rule deck to find the marker region in the SRAM array and then replicate the final via shift in the marker region across the SRAM array, without introducing MRC violations. See “[Example 8 — DFM Via Shift Marker Flow](#)” on page 295.

- **SPEC *spec_name* {*input_via_layer1* *input_via_layer2* | ('*input_via_layer_set*'')} *input_via_layer* | *input_via_layer* ('*input_via_layer_set*'') | ('*input_via_layer_set1*') ('*input_via_layer_set2*'')}**

An optional argument set referencing the *spec_name* and the *input_via_layer* and *input_via_layer_set* specifications. An *input_via_layer_set* can contain one or more different *input_via_layers* and ideally should be used for more than one layer. Multiple layers in an *input_via_layer_set* must be space-separated. The multi-layer SPEC argument prevents MRC violations between different *input_via_layer* types such as square and rectangular types. At least two different *input_via_layers* must be specified within the DFM Via Shift command to use the multi-layer SPEC keyword.

If both layer sets have more than one *input_via_layer*, the same layer can be present in both layer sets, but pairs formed between the same layers are ignored. If unmovable *input_via_layers* are specified in both layer sets with multiple layers, then pairs formed between the unmovable via layers in one set and the unmovable via layers in another set are ignored.

Every specified via layer should be provided as an *input_via_layer* to the operation. Any number of multi-layer SPEC keyword sets may be specified in a single DFM Via Shift command. See “[Example 2 — DFM Via Shift Using Multiple *input_via_layers*](#)” on page 288.

The following are examples of syntax usage:

- Specification between vias on *input_via_layer1* and *input_via_layer2*:
SPEC *spec_1* *input_via_layer1* *input_via_layer2*

- Specification between vias on input_via_layer1 and input_via_layer2, and vias on input_via_layer1 and input_via_layer3:

SPEC *spec_2* *input_via_layer1* (*input_via_layer2* *input_via_layer3*)

- Specification between vias on input_via_layer1 and input_via_layer3, and vias on input_via_layer2 and input_via_layer3:

SPEC *spec_3* (*input_via_layer1* *input_via_layer2*) *input_via_layer3*

- Specification between vias on input_via_layer1 and input_via_layer3, vias on input_via_layer1 and input_via_layer4, vias on input_via_layer2 and input_via_layer3, and vias on input_via_layer2 and input_via_layer4:

SPEC *spec_4* (*input_via_layer1* *input_via_layer2*) (*input_via_layer3* *input_via_layer4*)

- Specification between all pairs of vias on input_via_layer1, input_via_layer2, and input_via_layer3:

SPEC *spec_5* (*input_via_layer1* *input_via_layer2* *input_via_layer3*)
(*input_via_layer1* *input_via_layer2* *input_via_layer3*)

If a two-layer SPEC is defined between a movable via layer and an unmovable via layer, and the size of the unmovable via layer is not specified, then the suggested SPEC usage is either spec_4 or spec_5, depending on whether the movable via is square or rectangular, respectively. If the correct SPEC is not specified, then the missing MRC constraint values are deduced from the given SPEC.

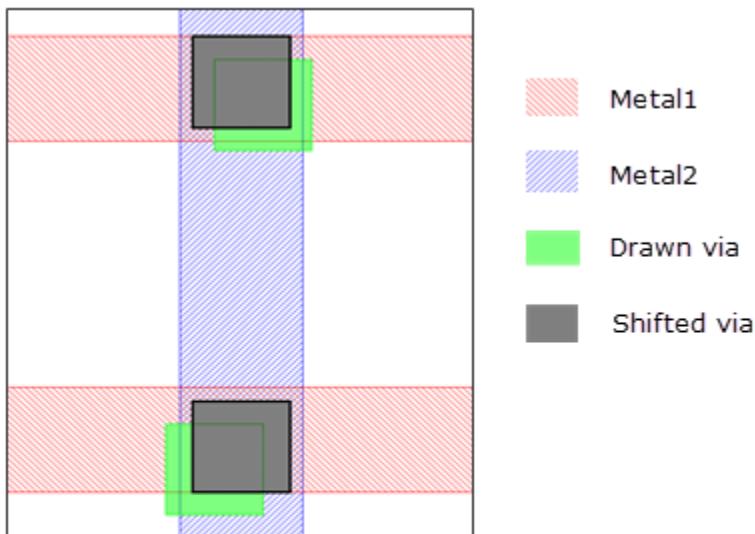
For example, if a movable via is square, and the provided SPEC is SQUARE, then the missing values for LONG_SQUARE and SHORT_SQUARE are the same as that of SQUARE. Similarly, if a movable via is rectangular, and the provided SPEC is LONG_SQUARE and SHORT_SQUARE, then the missing LONG_LONG, LONG_SHORT, and SHORT_SHORT values are deduced from the maximum of the values for LONG_SQUARE and SHORT_SQUARE.

See also “[Example 3 — DFM Via Shift Detection of Invalid SPEC Syntax](#)” on page 289.

Description

The DFM Via Shift operation moves vias to improve enclosure by the specified metal layers. MRC spacing constraints for the vias are specified by a [DFM Spec Via Shift](#) specification, which is referenced by DFM Via Shift.

DFM Via Shift is typically used after metal biasing and retargeting in order to improve via placement. The biasing and retargeting process often results in vias that are not fully enclosed by metal layers, which can reduce via yield. DFM Via Shift is used to move the vias without introducing MRC violations, as shown in the following figure.

Figure 11-2. Via Shifting

The metal layers shown have been biased and re-targetted, and subsequently do not cover drawn vias. DFM VIA SHIFT moves the vias back into an MRC-compliant position.

Multiple DFM Via Shift commands with different input layers can reference the same DFM Spec Via Shift specification. This makes it easy to define common MRC constraints and apply them to different via layers.

Each input_via_layer should have only vias of the same dimensions, and should not have any MRC violations. Only square and rectangular vias should be provided as input.

Via Shift Specification

The via_shift_args argument set defines the rules for shifting vias on one layer. You can specify more than one set of via_shift_args within a single DFM Via Shift command.

There are two use cases when shifting vias:

- **Non-Interacting Vias** — Shifting vias on the input_via_layer does not cause MRC violations on another via layer. In this case, only one via_shift_args argument set is specified. See “[Example 1 — DFM Via Shift Command](#)” on page 288.
- **Interacting Vias** — Shifting vias on the input_via_layer can cause MRC violations on another via layer. In this case, a via_shift_args argument set is specified for each interacting via layer that is to be shifted. The DFM Via Shift operation optimizes the via movement for each via layer while taking the MRC constraints for each layer into account. You can also provide a multi-layer SPEC argument set to define the MRCs between the interacting via layers. See “[Example 2 — DFM Via Shift Using Multiple input_via_layers](#)” on page 288.

A shifted via layer and an unmovable via layer (`max_shift=0`) fall in this category if the shifted vias can cause MRC violations between the two via layers.

Examples

Example 1 — DFM Via Shift Command

This example shifts vias on the via layer. Vias on this layer do not cause MRC violations on any other via layer, so only one `input_via_layer` is specified. The DFM Spec Via Shift statement name is highlighted in green.

```
DFM SPEC VIA SHIFT spec_1
    SQUARE OPPOSITE 0.08225 EXTEND BY 0.0235 SPACE 0.0725
    shifted_via = DFM VIA SHIFT
        [via(VIA 0.005) m_low(LOWER) m_up(UPPER) SPEC spec_1]
```

Example 2 — DFM Via Shift Using Multiple input_via_layers

This example shifts the vias on layers `sq_via` and `bar_via`, where movement of the vias on one layer can cause MRC violations on the other layer. The specified commands and keywords perform the following actions:

- The DFM Spec Via Shift command first defines three SPECs, `sq2sq`, `bar2bar`, and `bar2sq`.
- The DFM Via Shift command then specifies two `input_via_layers`, `sq_via` and `bar_via`, and applies their respective SPEC names, enforcing individual constraints.
- The multi-layer SPEC argument applies another SPEC name enforcing separate constraints between the different vias, `sq_via` and `bar_via`. The various SPEC names are highlighted in consistent colors.

```
// Variable values
VARIABLE Sq2Sq_OPP 0.060
VARIABLE Sq2Sq_EXT 0.020
VARIABLE Sq2Sq_SPACE 0.055
VARIABLE Bar_OPP 0.070
VARIABLE Bar_EXT 0.025
VARIABLE Bar_SPACE 0.060
VARIABLE MAX_SHIFT 0.010

// Intermediate layer derivations
sq_via = RECTANGLE via ASPECT == 1
bar_via = RECTANGLE via ASPECT == 2

DFM SPEC VIA SHIFT sq2sq
    SQUARE OPPOSITE Sq2Sq_OPP EXTEND BY Sq2Sq_EXT SPACE Sq2Sq_SPACE
DFM SPEC VIA SHIFT bar2bar
    LONG_LONG OPPOSITE Bar_OPP EXTEND BY Bar_EXT SPACE Bar_SPACE
    LONG_SHORT OPPOSITE Bar_OPP EXTEND BY Bar_EXT SPACE Bar_SPACE
    SHORT_SHORT OPPOSITE Bar_OPP EXTEND BY Bar_EXT SPACE Bar_SPACE
DFM SPEC VIA SHIFT bar2sq
    LONG_SQUARE OPPOSITE Bar_OPP EXTEND BY Bar_EXT SPACE Bar_SPACE
    SHORT_SQUARE OPPOSITE Bar_OPP EXTEND BY Bar_EXT SPACE Bar_SPACE
```

```

shifted_via = DFM VIA SHIFT
  [sq_via(VIA MAX_SHIFT)
    m1(LOWER) m2(UPPER) exclusion(EXCLUSION) SPEC sq2sq]
  [bar_via(VIA MAX_SHIFT)
    m1(LOWER) m2(UPPER) exclusion(EXCLUSION) SPEC bar2bar]
SPEC bar2sq sq_via bar_via

```

Example 3 — DFM Via Shift Detection of Invalid SPEC Syntax

The DFM Via Shift operation detects and reports scenarios during rules compilation where vias are not defined properly (or undefined) in a multi-layer SPEC argument. Invalid SPEC syntax can cause uncertainty in via types (square or rectangular) and result in MRC inconsistencies.

For example, given DFM Spec Via Shift definitions as shown,

```

DFM SPEC VIA SHIFT spec_1
  SQUARE OPPOSITE 0.08225 EXTEND BY 0.0235 SPACE 0.0725
DFM SPEC VIA SHIFT spec_2
  LONG_LONG OPPOSITE 0.06625 EXTEND BY 0.033 SPACE 0.0485
  SHORT_SHORT OPPOSITE 0.06625 EXTEND BY 0.033 SPACE 0.0485
  LONG_SHORT OPPOSITE 0.06625 EXTEND BY 0.033 SPACE 0.033
DFM SPEC VIA SHIFT spec_3
  LONG_SQUARE OPPOSITE 0.06625 EXTEND BY 0.033 SPACE 0.0485
  SHORT_SQUARE OPPOSITE 0.06625 EXTEND BY 0.033 SPACE 0.0485

```

the following examples are reported as invalid syntax:

- Invalid single-layer SPEC.
 - Example: spec_3 checks constraints between square versus rectangular vias and cannot be a single-layer SPEC.

```

shifted_via = DFM VIA SHIFT
  // square or slot via type?
  [via_1(VIA 0.004) m_low(LOWER) m_up(UPPER) SPEC spec_3]

```

- Invalid 2-layer SPEC.
 - Example: spec_1 checks constraints between square versus square vias and cannot check constraints between different via types (via_1 is square type, via_2 is rectangular type).

```

shifted_via = DFM VIA SHIFT
  // square type
  [via_1(VIA 0.004) m_low(LOWER) m_up(UPPER) SPEC spec_1]
  // rectangular type
  [via_2(VIA 0.004) m_low(LOWER) m_up(UPPER) SPEC spec_2]
SPEC spec_1 via_1 via_2

```

- Cannot define SPEC between unmovable layers.
 - Example: spec_1 checks constraints between two unmovable layers.

```
shifted_via = DFM VIA SHIFT
    // square type
    [via_1(VIA 0.004) m_low(LOWER) m_up(UPPER) SPEC spec_1]
    [via_3(VIA 0.0)]
    [via_4(VIA 0.0)]
SPEC spec_1 via_3 via_4 // problematic statement
```

Example 4 — Layers for Input Layout Analysis

The DFM Via Shift operation filters vias of invalid size (not equal to SIZE) and shape (neither square nor rectangular) on the input layer during the run. These invalid vias are unmovable and do not participate in the shift operation, but they appear on the output layer along with the valid vias. When certain map_option arguments are used, the functionality outputs the invalid vias to specified output layers that can be used for input layer analysis.

For example, given DFM Spec Via Shift definitions as shown,

```
DFM SPEC VIA SHIFT spec_1
    SQUARE OPPOSITE 0.08225 EXTEND BY 0.0235 SPACE 0.0725
```

when the via parameters are the same, these commands run concurrently and output the shifted via layer along with the invalid via layers.

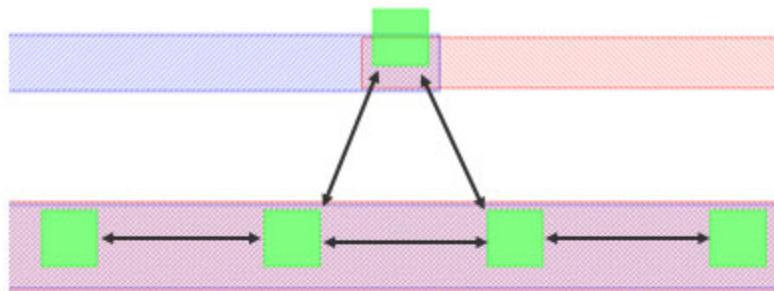
```
shifted_via = DFM VIA SHIFT
    [via_sq(VIA 0.010 SIZE 0.040) cm_sq(COMMON) SPEC spec_1]
invalid_size_via = DFM VIA SHIFT INVALID_VIA_SIZE
    [via_sq(VIA 0.010 SIZE 0.040) cm_sq(COMMON) SPEC spec_1]
invalid_shape_via = DFM VIA SHIFT INVALID_VIA_SHAPE
    [via_sq(VIA 0.010 SIZE 0.040) cm_sq(COMMON) SPEC spec_1]
```

The map_option arguments can be used alone (with the required DFM Via Shift arguments) to generate only the invalid via layers (no shifted via layer) for input layout analysis.

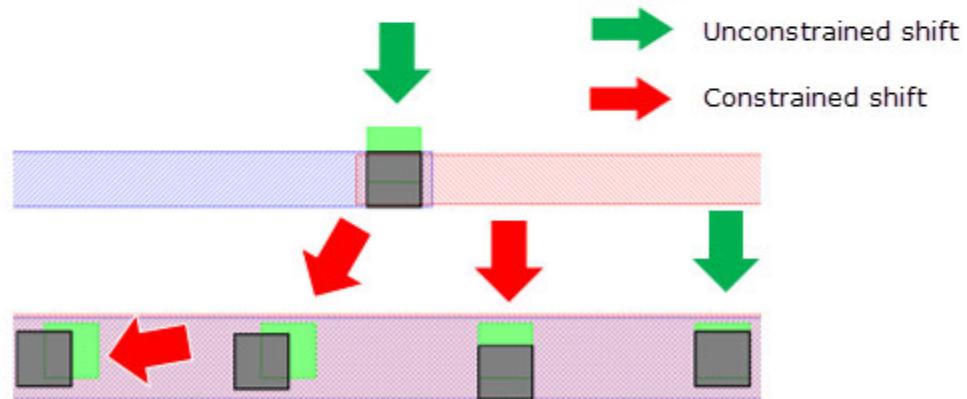
The INVALID_VIA_MIN_SPACING keyword option can be used with DFM Via Shift to specify the minimum spacing of movable vias with invalid vias.

Example 5 — Via Cluster Constraints and Optimization

A via cluster is a group of vias affecting each other during via shifting, due to the application of MRC constraints and the maximum shift value specified. DFM Via Shift functionality optimizes the entire cluster providing holistic coverage, providing priority for critical vias. This algorithm may result in minor placement degradation of less critical vias as diagrammed in [Figure 11-3](#).

Figure 11-3. DFM Via Shift Optimization

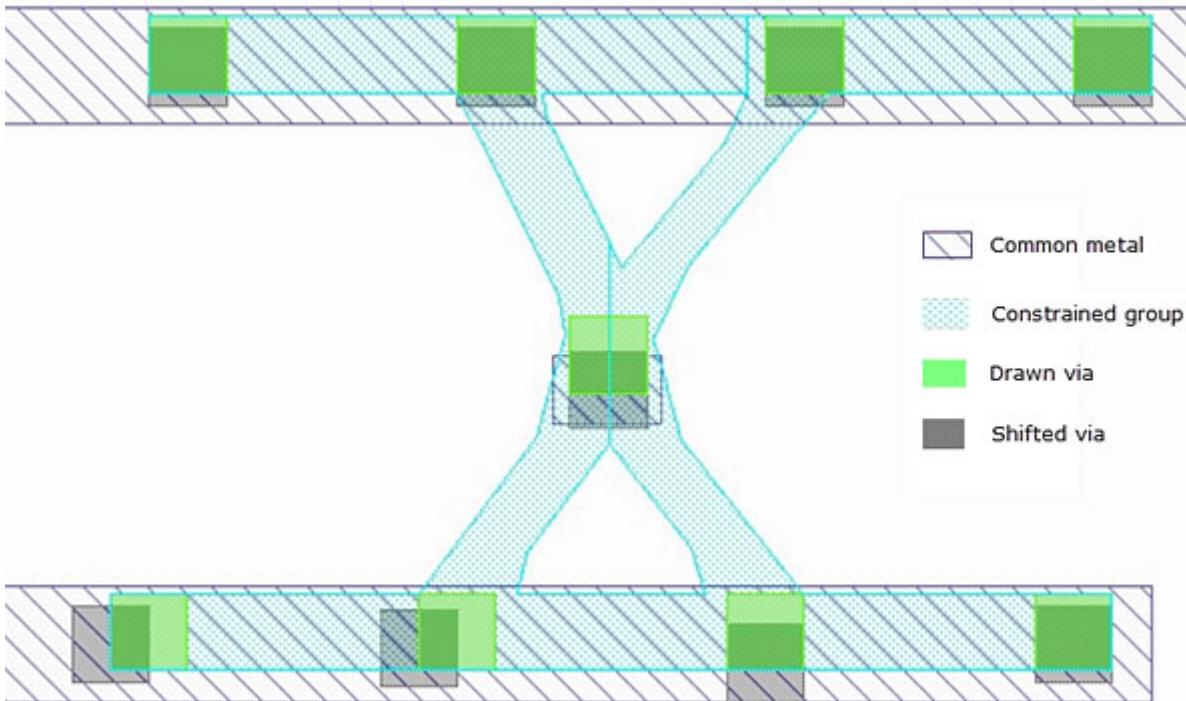
Vias constrained from moving by MRC rules result in clusters that must be co-optimized.



The enclosure of some vias may require minor degradation to optimize the location of more critical vias.

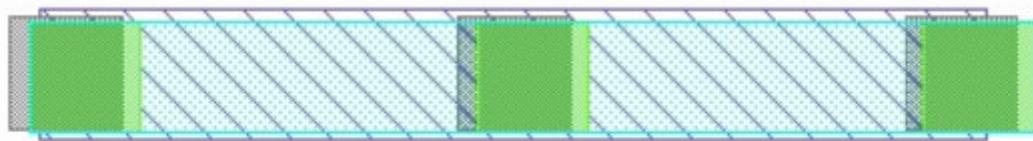
Clusters in this design context are represented here in [Figure 11-4](#). The aqua color represents the identification of the constraint area where MRC-related trade-offs may take place.

Figure 11-4. Constrained Via Clusters

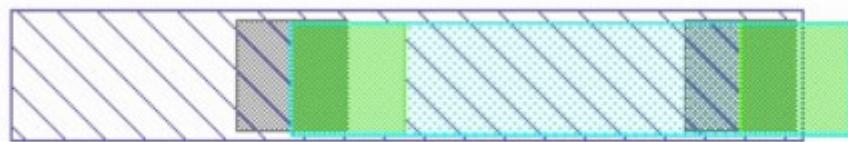


Other constraint-bound clusters result in balanced and improved coverage while maintaining MRC requirements as seen in these two examples, respectively, in [Figure 11-5](#).

Figure 11-5. Balanced and Increased Coverage Within Via Clusters



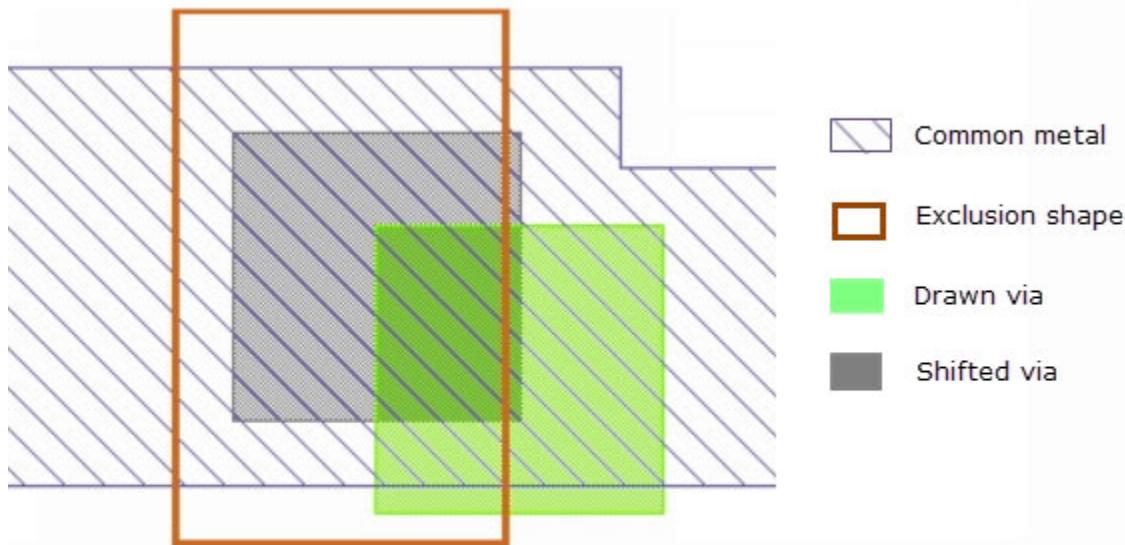
DFM VIA SHIFT balances via coverage where MRC constraints exist over common metal.



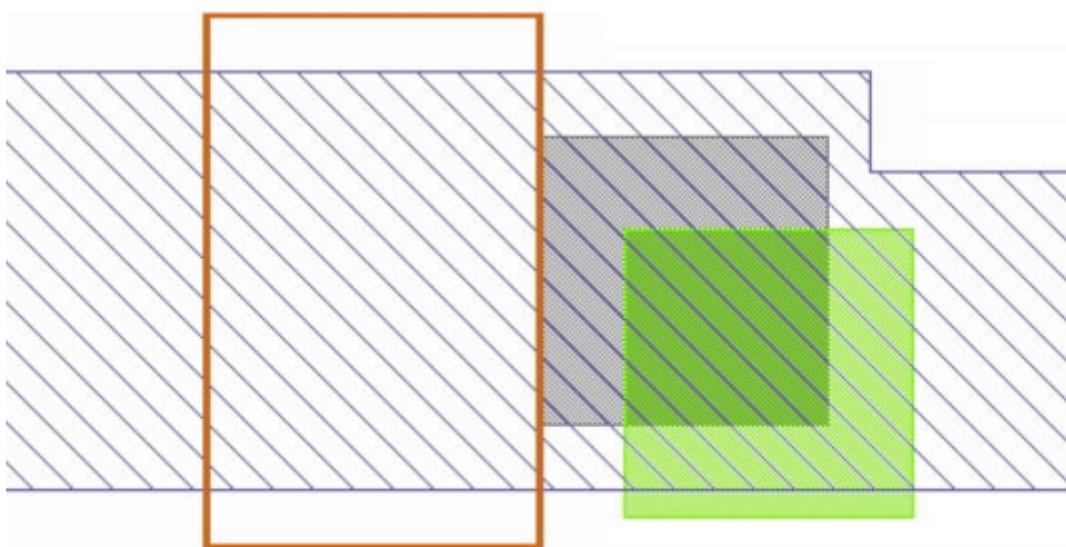
DFM VIA SHIFT ensures optimal or complete via coverage where MRC constraints exist over common metal.

Example 6 — Exclusion Shape Overlap With Vias

The function provided by the exclusion layer may or may not have an effect on vias that are drawn as overlapping with an exclusion shape. Any via drawn outside the same exclusion shape does honor that same exclusion shape boundary. [Figure 11-6](#) demonstrates a shifted via failing to honor the exclusion shape boundary (overlapping the drawn via), and [Figure 11-7](#) shows a functional use of the exclusion layer.

Figure 11-6. Exclusion Shape Overlapping With Drawn Via

Where drawn via shapes overlap an exclusion shape, their resulting shifted via shapes do not comply with the same exclusion shape.

Figure 11-7. Functional Exclusion Shape Usage

A properly utilized exclusion shape constrains shifted via movement to its boundary.

Example 7 — Via Edge Resize for MRC Constraints with EXTEND BY

This example uses DFM Via Shift with the EXTEND BY keyword option to shift vias on the input layers and apply MRC constraints to resized edges of the shifted vias. The resized via edges are only used for applying MRC constraints and do not affect the size of the output via. This type of specification is useful when different targeted via sizes and MRC constraints are needed during the wafer process flow. The EXTEND BY keyword option can be used to set two size parameters for each input via layer, one on the short edge and another on the long edge. If the input layer is a square via, only the short edge parameter is used. The values for EXTEND BY can be positive (extends both ends of an edge), negative (retracts both ends of an edge), or zero (no resizing of an edge). The specified commands and keywords perform the following actions:

- The DFM Spec Via Shift command first defines three SPECs, spec_1, spec_2, and spec_3. The various SPEC names are highlighted in consistent colors.
- The DFM Via Shift command then specifies two input_via_layers, via_sq and via_rect, and applies their respective SPEC names, enforcing individual constraints.
- The EXTEND BY keyword extends the edges of shifted vias by the specified parameters. In the example, both ends of the short edges are extended for the square via. For the rectangular via, both ends of the long edges are extended; the short edge is unchanged because its specification is zero.
- The multi-layer SPEC argument applies another SPEC name enforcing separate constraints between the different input_via_layers via_sq and via_rect.

```
// Variable values
VARIABLE MAXS 0.0100
VARIABLE SHORT_SIDE 0.0235
VARIABLE OPP 0.0723
VARIABLE EXT 0.0115
VARIABLE SPC 0.0620
VARIABLE EXTEND_SHORT_BY 0.0050
VARIABLE EXTEND_LONG_BY 0.0025

// Intermediate layer derivations
via_sq = RECTANGLE via ASPECT == 1
via_rect = RECTANGLE via ASPECT == 2

DFM SPEC VIA SHIFT spec_1
    SQUARE OPPOSITE OPP EXTEND BY EXT SPACE SPC
DFM SPEC VIA SHIFT spec_2
    LONG_LONG OPPOSITE OPP EXTEND BY EXT SPACE SPC
    LONG_SHORT OPPOSITE OPP EXTEND BY EXT SPACE SPC
    SHORT_SHORT OPPOSITE OPP EXTEND BY EXT SPACE SPC
DFM SPEC VIA SHIFT spec_3
    LONG_SQUARE OPPOSITE OPP EXTEND BY EXT SPACE SPC
    SHORT_SQUARE OPPOSITE OPP EXTEND BY EXT SPACE SPC
```

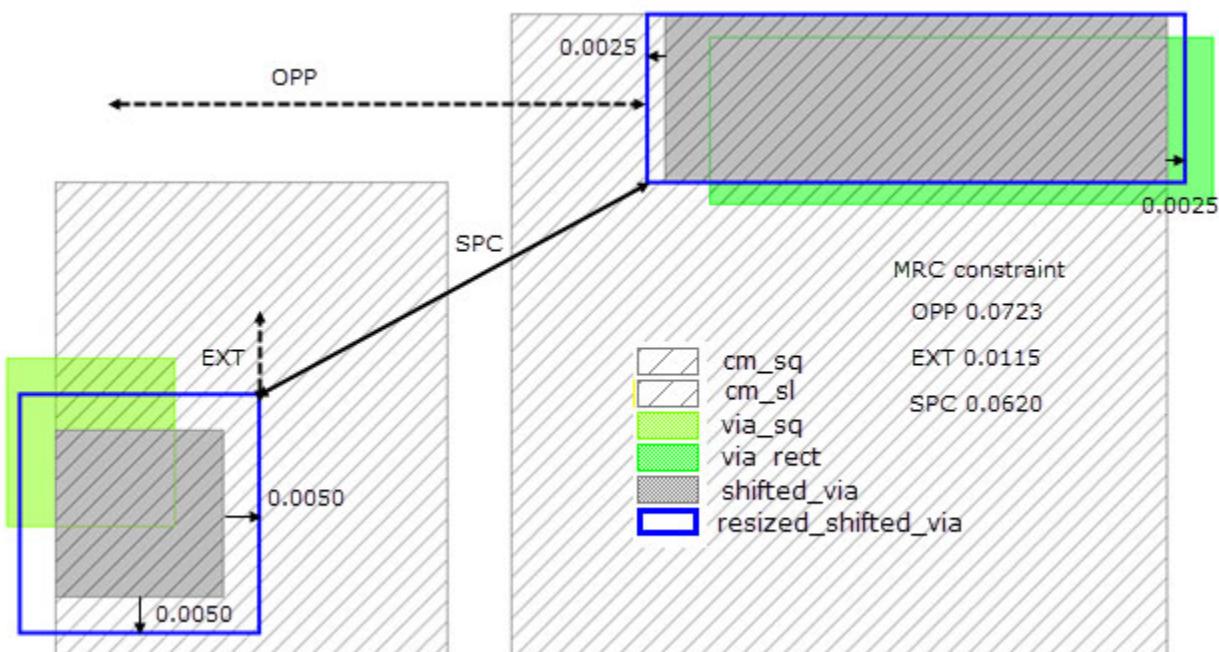
```

shifted_via = DFM VIA SHIFT
  [via_sq(VIA MAXS EXTEND_BY EXTEND_SHORT_BY) cm_sq(COMMON)
   SPEC spec_1]
  [via_rect(VIA MAXS EXTEND_BY 0::EXTEND_LONG_BY) cm_sl(COMMON)
   SPEC spec_2]
  SPEC spec_3 via_rect via_sq

```

Figure 11-8 shows DFM Via Shift EXTEND BY expansions of the short and long edges of the respective shifted vias for via_sq and via_rect. These dimensional changes are only for measuring the MRC constraint (the size of the output shifted_via is unchanged). The MRC constraint is measured between the resized_shifted_vias based on the corner-to-corner Euclidean spacing value in spec_3 (SPACE SPC) and defined in the variable statement (VARIABLE SPC 0.0620). Relevant MRC constraints for (OPPOSITE OPP) and (EXTEND BY EXT) in spec_3 are also shown. See DFM Spec Via Shift “[MRC Zones for Vias](#)” on page 278 for definitions of the MRC constraints. All dimensions are in microns.

Figure 11-8. DFM Via Shift EXTEND BY



Example 8 — DFM Via Shift Marker Flow

This example uses DFM Via Shift with the marker flow to ensure via shift results are the same across an entire SRAM array. The following figures show the stages of the flow starting with the input array, the marker region detail, and the replicated marker region across the array.

Figure 11-9. DFM Via Shift Input Array With Marker Region

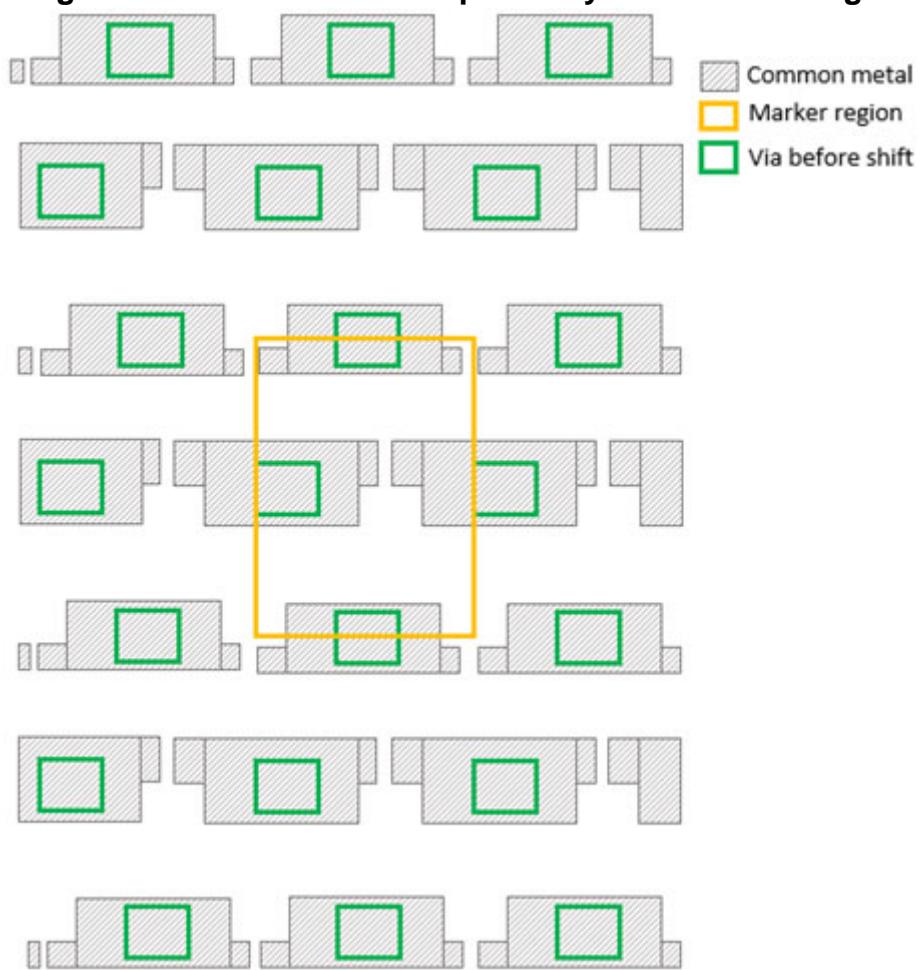


Figure 11-10. DFM Via Shift Marker Region Detail

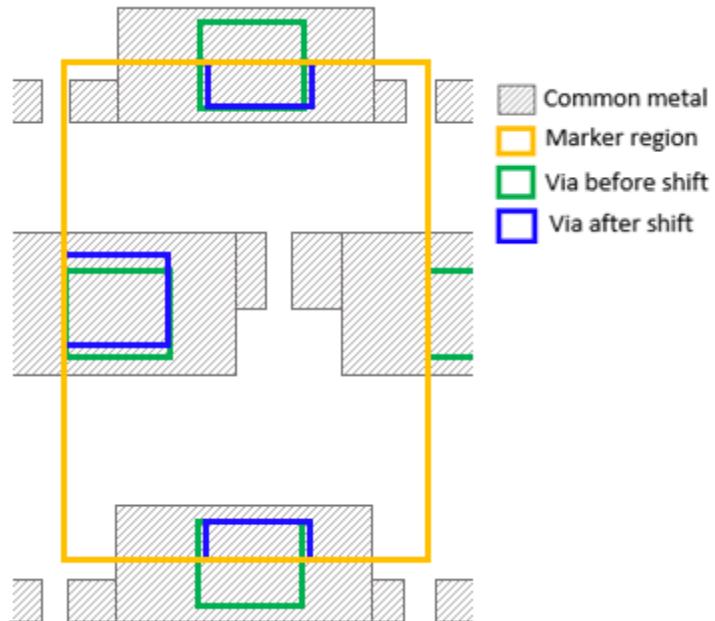
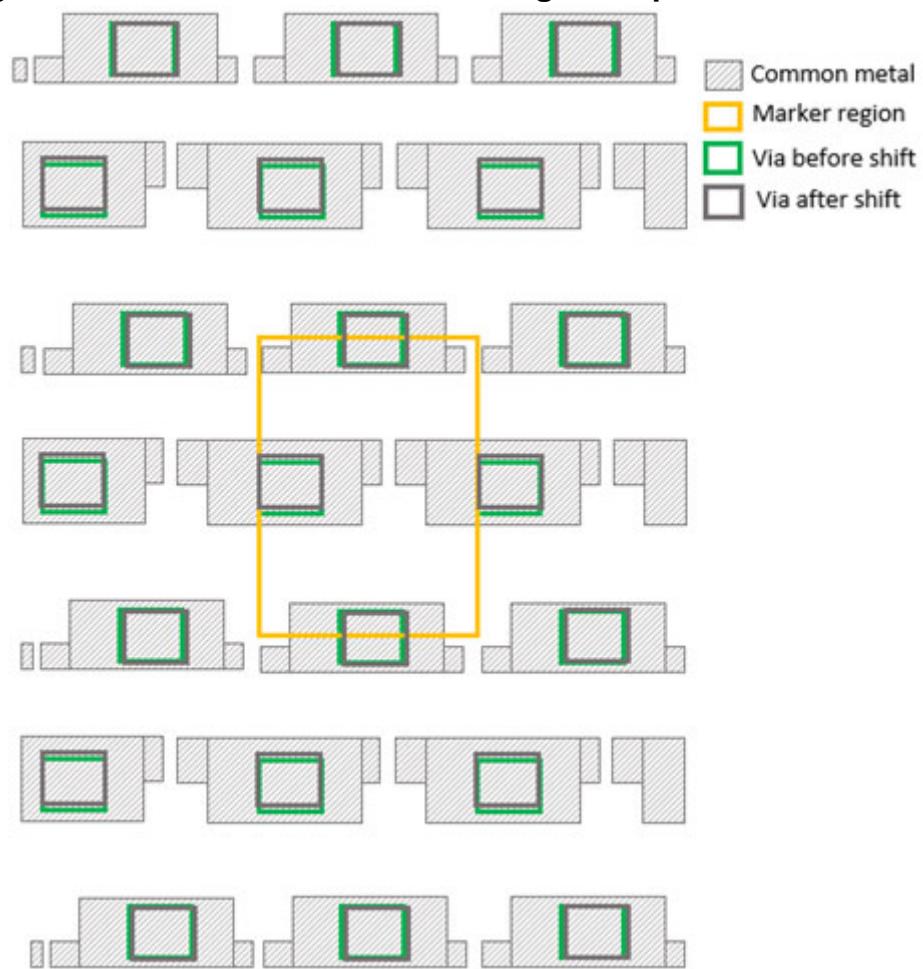


Figure 11-11. DFM Via Shift Marker Region Replicated Across Array



DFM Reshape Commands

The reference information for the Calibre DFM Reshape commands is provided. These commands use a Calibre YieldEnhancer license.

For more information on Calibre YieldEnhancer via applications, see also “[DFM Via Shift Commands](#)” on page 274 and “[Layout Enhancement](#)” on page 81.

DFM Spec Reshape	299
DFM Reshape.....	303

DFM Spec Reshape

SVRF commands: Specification statement

Specifies a set of correction values that are used to transform corner and line-end metal shapes for improved via enclosure. Referenced by DFM Reshape. Uses a Calibre YieldEnhancer license.

Usage

Usage for Corner Transformations

DFM SPEC RESHAPE

spec_name
CORNER {*leg_value*‘(*length_value*)’} ...

Usage for Line-End Transformations

DFM SPEC RESHAPE

spec_name
LINE_END {{*leg_value*[::*width_adjustment_value*]}} | {*leg_value*[‘(*width_value*)’]} ... }

Arguments

- ***spec_name***

A required argument labeling the specifications for corner and line-end shape transformations by a subsequent DFM Reshape command. Each DFM Spec Reshape spec_name must be unique. Multiple DFM Reshape commands with different input layers may share a single DFM Spec Reshape statement.

- **CORNER** {*leg_value*‘(*length_value*)’} ...

A required keyword and argument set specifying corner transformation shapes of the enclosing metal. The result of this specification is an isosceles triangle that defines a corner shape based on the shape of the actual metal. The values are positive floating-point numbers in user units. More than one set of space-separated “*leg_value*(*length_value*)” pairs can optionally be specified.

leg_value — Required argument that specifies the length of the legs of the result triangle.

‘(*length_value*)’ — Required argument, enclosed in parentheses, specifying that a corner with both edges greater than this value generates the result triangle.

- **LINE-END** {{*leg_value*[::*width_adjustment_value*]}} | {*leg_value*[‘(*width_value*)’]} ... }

A required keyword and argument set specifying line-end transformation shapes of the enclosing metal. The result is a rectangle based on the shape of the actual metal with length (perpendicular to the line-end edge) and width (parallel to the line-end edge). The values are floating-point numbers in user units. More than one set of space-separated “*leg_value*(*width_value*)” pairs can optionally be specified.

leg_value — Required argument specifying the length of the result rectangle. The value can be a positive or negative floating-point number. Positive and negative values affect the rectangle generation as follows:

- When the *leg_value* is positive, the rectangle is generated outward from the line-end edge (extension rectangle).
- When the *leg_value* is negative, the rectangle is generated inward from the line-end edge (pullback rectangle).

When the input layer RESIZE attribute is specified in the DFM RESHAPE statement, the line_end rectangle is applied to the resized line-end.

width_adjustment_value — Optional argument specifying a width adjustment value added to both ends of the width dimension of the result rectangle (line-end width plus 2* *width_adjustment_value*). The default *width_adjustment_value* is 0.

('width_value') — Optional argument, enclosed in parentheses, specifying that result rectangles are generated for line-end edges with width greater than or equal to this value. The default *width_value* is equal to the line-end width.

Description

The DFM Spec Reshape statement specifies the values used by a [DFM Reshape](#) operation to improve corner and line-end metal enclosures for vias. The corner shapes are triangles with edge pairs at convex corners (corner-chop) and concave corners (corner-fill). Line-end shapes are defined as rectangles.

DFM Spec Reshape and DFM Reshape are used together to process the layers before a DFM Via Shift operation. See “[DFM Via Shift Commands](#)” on page 274.

Examples

Example 1

This example generates corner-chop triangles with length 0.014 at a convex corner, when both edges are longer than 0.022, and corner-chop triangles with length 0.021, when both edges are longer than 0.033. The DFM Spec Reshape spec_name is referenced by a DFM RESHAPE CORNER_CONVEX statement. See “[Case 2: CORNER \(CORNER_CONVEX Edge Pairs\)](#)” on page 310 for the example figure.

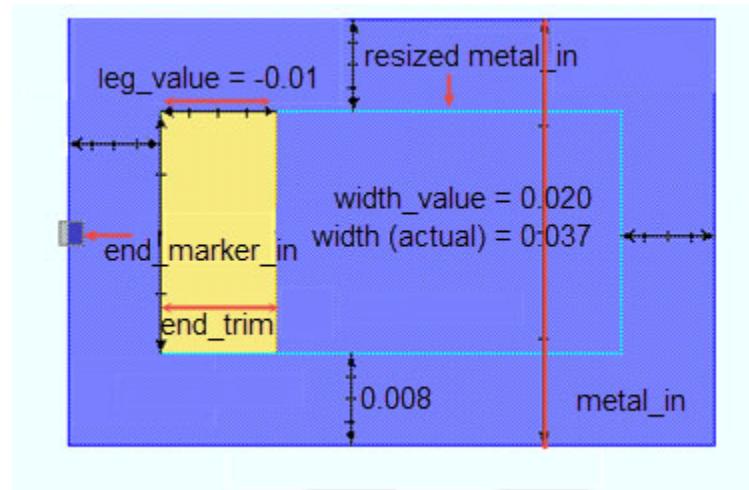
```
DFM SPEC RESHAPE
    corner_chop_spec CORNER 0.014(0.022) 0.021(0.033)
    low_metal_chop = DFM RESHAPE
        CORNER_CONVEX(CONVEX_SPEC corner_chop_spec) m_low
```

Example 2

This example generates a trim-rectangle on a resized input layer. The rectangle is generated inward from the line-end of the resized layer with a leg_value of -0.01 when the line-end width is $\geq 0.020 < 0.050$. Because the RESIZE value is negative, the width of the rectangle is the line-end width of the input layer decreased by 2* RESIZE value. The DFM Spec Reshape spec_name is referenced by a DFM RESHAPE LINE_END_PULLBACK statement.

```
DFM SPEC RESHAPE line_end_spec LINE_END -0.010(0.020) 0.015(0.050)
end_trim = DFM RESHAPE LINE_END_PULLBACK(LINE_END_SPEC line_end_spec)
    metal_in(RESIZE (-0.008)) via_reshape(MASK 0.05)
end_marker_in(LINE_END)
```

Figure 11-12. DFM Spec Reshape LINE_END With DFM RESHAPE LINE_END_PULLBACK

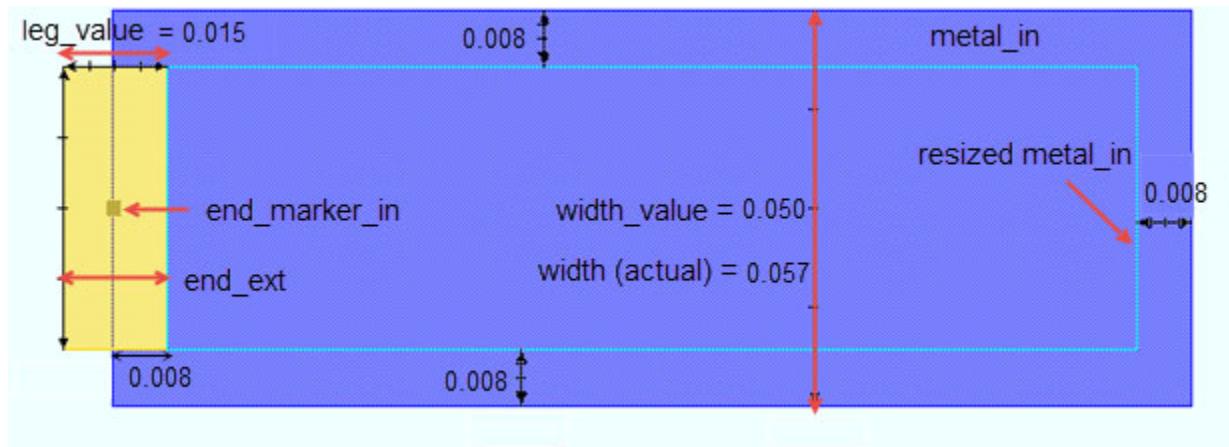


Example 3

This example generates an extension-rectangle on a resized input layer. The rectangle is generated outward from the line-end edge of the resized layer with a leg_value of 0.015 when the line-end width is ≥ 0.050 . Because the RESIZE value is negative, the width of the rectangle is the line-end width of the input layer *decreased* by $2 * \text{RESIZE}$ value. The DFM Spec Reshape spec_name is referenced by a DFM RESHAPE LINE_END_EXTENSION statement.

```
DFM SPEC RESHAPE line_end_spec LINE_END -0.010(0.020) 0.015(0.050)
end_ext = DFM RESHAPE LINE_END_EXTENSION(LINE_END_SPEC line_end_spec)
    metal_in(RESIZE (-0.008)) via_reshape(MASK 0.05)
    end_marker_in(LINE_END)
```

Figure 11-13. DFM Spec Reshape LINE_END With DFM RESHAPE LINE_END_EXTENSION



DFM Reshape

Performs geometric shape transformations on the input layer to imitate actual metal shapes for subsequent via operations. Requires DFM Spec Reshape for constraint specifications. Uses a Calibre YieldEnhancer license.

Usage

```
output_layer = DFM RESHAPE
  transformation[('transformation_options...')]
  input_layer[('input_attributes')]
  [selection_layer('{'MARKER aoi_value | MASK [aoi_value]}')]
  [line_end_layer('LINE_END')]
```

Arguments

- **output_layer**
A required output layer that can be saved to the output database.
- **transformation[('transformation_options...')]**
A required argument specifying the geometry transformation. A transformation_options specification may be required according to the type of transformation used.

The following geometry transformations are supported in hierarchical and flat execution using SVRF or embedded SVRF rules:

CORNER_CONCAVE — Generates triangles at concave vertices (corner-fill) and optionally, rectangles at line-ends of the input layer. The triangles are generated by a DFM SPEC RESHAPE CORNER specification and used for corner correction. The rectangles are generated by a DFM SPEC RESHAPE LINE_END specification and used for line-end correction to the edges touched by the line-end layer.

At least one corresponding specification (CONCAVE_SPEC or LINE_END_SPEC) for the transformation type must be specified to generate results. For run concurrency, all specifications (CONCAVE_SPEC, CONVEX_SPEC, and LINE_END_SPEC) are compatible. For example, CORNER_CONCAVE can be specified with LINE_END_SPEC as follows:

```
DFM RESHAPE CORNER_CONCAVE (CONCAVE_SPEC corner_fill_spec
CONVEX_SPEC corner_chop_spec LINE_END_SPEC line_end_spec) ...
```

The resulting transformation produces corner fill and line-end extension geometries together on the same output layer. See “[DFM Spec Reshape](#)” on page 299 for LINE_END extension definition.

RESIZE is the only supported input layer attribute.

CORNER_CONVEX — Generates triangles at convex vertices (corner-chop) and optionally, rectangles at line-ends of the input layer. The triangles are generated by a DFM SPEC RESHAPE CORNER specification and used for corner correction. The rectangles are generated by a DFM SPEC RESHAPE LINE_END specification and used for line-end correction to the edges touched by the line-end layer.

At least one corresponding specification (CONVEX_SPEC or LINE_END_SPEC) for the transformation type must be specified to generate results. For run concurrency, all specifications (CONCAVE_SPEC, CONVEX_SPEC, and LINE_END_SPEC) are compatible. For example, CORNER_CONVEX can be specified with LINE_END_SPEC as follows:

```
DFM RESHAPE CORNER_CONVEX (CONCAVE_SPEC corner_fill_spec CONVEX_SPEC corner_chop_spec LINE_END_SPEC line_end_spec) ...
```

The resulting transformation produces corner-chop and line-end pullback geometries together on the same output layer. See “[DFM Spec Reshape](#)” on page 299 for LINE-END pullback definition.

RESIZE is the only supported input layer attribute.

UNIFORM — Selects geometries of the specified size from the input layer. The output layer contains only these geometries, which may be resized. This transformation requires the BY RECTANGLE input attribute and can optionally be specified with GRID or FIXED. The input dimensions for the selected geometries can vary if the input_layer TOLERANCE attribute is specified. The output is resized in these cases:

- TOLERANCE specified without new_dimension1 or new_dimension2 — The output shapes have uniform dimensions of dimension1 and dimension2.
- new_dimension1 and/or new_dimension2 arguments specified — The output shapes have the dimensions new_dimension1 and/or new_dimension2.

The UNIFORM transformation cannot be used with the selection_layer and line_end_layer options. Run concurrency is supported when the input layers are the same, and if specified, the RESIZE input layer attributes are the same.

LINE_END_PULLBACK — Generates result rectangles from line-end edges (polygon edges that interact with line_end_layer) of the input layer. The output rectangles (pullback rectangles) are generated inward from the line-end edge of the resized input layer. The shrink distance of the line-end is defined by the leg_value in the DFM SPEC RESHAPE LINE-END specification. A negative leg_value generates pullback rectangles. At least one line-end spec_name must be specified from transformation_options. See “[DFM Spec Reshape LINE-END With DFM RESHAPE LINE-END_PULLBACK](#)” on page 301. RESIZE is the only supported input layer attribute.

LINE_END_EXTENSION — Generates result rectangles from line-end edges (polygon edges that interact with line_end_layer) of the input layer. The output rectangles (extension rectangles) are generated outward from the line-end edge of the resized input layer. The extension distance of the line-end is defined by the leg_value in the DFM SPEC RESHAPE LINE-END specification. A positive leg_value generates extension rectangles. At least one line-end spec_name must be specified from transformation_options. See “[DFM Spec Reshape LINE-END With DFM RESHAPE LINE-END_EXTENSION](#)” on page 302. RESIZE is the only supported input layer attribute.

LINE_END_ALL — Outputs the resized input layer with line-end transformations applied to it. At least one LINE_END_SPEC must be specified in the transformation_options. RESIZE is the only supported input layer attribute.

The line-end transformations are applied as follows:

- Line-end extensions are applied using an OR operation with the resized input layer.
- Line-end pullbacks are applied using a NOT operation with the resized input layer.

In terms of layer operations, the output layer is defined as follows:

```
out_layer = ((resize_input OR line_end_ext) NOT line_end_pb)
```

'(transformation_options ...)' — An optional or required argument according to the transformation. More than one spec_name may be specified for the transformation option (concave, convex, and line-end).

CONCAVE_SPEC spec_name — Specifies a DFM SPEC RESHAPE CORNER specification for generating result triangles (corner-fill) on concave corners. At least one transformation spec_name must be specified. Compatible with CONVEX_SPEC and LINE_END_SPEC specifications.

CONVEX_SPEC spec_name — Specifies a DFM SPEC RESHAPE CORNER specification for generating result triangles (corner-chop) on convex corners. At least one transformation spec_name must be specified. Compatible with CONCAVE_SPEC and LINE_END_SPEC specifications.

LINE_END_SPEC spec_name — Specifies a DFM SPEC RESHAPE LINE_END spec_name for generating result rectangles at line-ends. At least one LINE_END_SPEC spec_name must be specified for line-end transformations. For run concurrency, LINE_END_SPEC is compatible with CONCAVE_SPEC and CONVEX_SPEC specifications when the transformation type is CORNER_CONCAVE or CORNER_CONVEX.

See “[Example 1 — DFM RESHAPE CORNER Specifications](#)” on page 309, “[Example 2 — DFM RESHAPE UNIFORM Specifications](#)” on page 311, and “[Example 3 — DFM RESHAPE LINE-END Specifications](#)” on page 314.

- **input_layer['(input_attributes)']**

A required argument specifying the input layer to be transformed. The input layer can be specified with one or more attribute keywords in parentheses. Attribute values are positive floating-point numbers in user units, unless stated otherwise. The following input_attributes are listed according to the transformation:

RESIZE resize_value

Resizes the input layer by the specified value as a part of the transformation. The resize value can be a positive or negative floating-point number. The corner-chop and corner-fill triangles and line-end rectangles are applied to the resized input layer.

BY RECTANGLE *dimension1[::new_dimension1] [dimension2[::new_dimension2]]*
[TOLERANCE *tolerance_value*]

Selects Manhattan rectangles with sides equal to dimension1 and dimension2 (in horizontal or vertical orientation) from the input layer. This attribute is required for the UNIFORM transformation and can *only* be specified with the UNIFORM transformation. The BY RECTANGLE attribute can be specified with the GRID attribute and the FIXED attribute.

The following behavior applies:

- If only dimension1 is specified, square geometry shapes of that size are selected.
- If new dimensions are *not* specified, the output rectangles have the dimensions given by dimension1 and dimension2.
- If new dimensions are specified, the output rectangles have those dimensions instead and are centered in the original shape.
- If TOLERANCE is specified, this causes a resizing of the input shapes to uniform dimensions. The dimensions of the input shape can differ from the specified values by up to the tolerance_value (absolute tolerance).

FIXED

Applies to the UNIFORM transformation. This attribute identifies the orientation (horizontal or vertical) of Manhattan rectangles from the top cell view by checking the BY RECTANGLE values. If dimension1 is greater than dimension2, then the transformation only selects rectangles placed along the horizontal axis, otherwise, it selects rectangles placed along the vertical axis. Both dimensions must be specified. If the FIXED attribute is not specified, the transformation outputs rectangles in both horizontal and vertical orientations.

GRID *precision_value*

Places the output rectangles on a database grid specified by precision_value instead of the *database precision*. The operation places the lower left vertex of the output shape at a valid grid location, but does not adjust the dimensions of the output shape. If the specified dimensions of the output shape are not integer multiples of the grid spacing, only the lower left corner is on grid. For example, if the rule file (and the database) precision is 8000, but the DFM Reshape operation specifies GRID 4000, then the rectangle vertices can only be placed on the even vertices of the database grid. The precision_value is specified as a positive integer. See “[Precision](#)” in the *Standard Verification Rule Format (SVRF) Manual* for more information. The GRID attribute can be specified with the BY RECTANGLE attribute.

- *selection_layer*‘({MARKER *aoi_value* | MASK [*aoi_value*]})’

An optional argument specifying a polygon layer that defines areas of interest where the transformation is applied. The selection layer can be specified for either markers or regions using the MARKER or MASK keywords, respectively. The aoi_value is used to further define the area of interest.

‘(‘**MARKER** *aoi_value*‘)’ — Defines the areas of interest as equally sized squares (of *aoi_value*) centered on the marker polygons on the selection layer. For non-rectangular polygons, the center of the rectangular extent is used.

‘(‘**MASK** [*aoi_value*]‘)’ — Defines the areas of interest as rectangles equal to the extents of the mask polygons on the selection layer. The areas of interest can optionally be oversized by the *aoi_value*, similar to specifying **SIZE** *selection_layer* BY *aoi_value*.

aoi_value

A numerical value specifying the area of interest on the selection layer used with **MARKER** or optionally, **MASK** keywords. The value is a positive floating-point number in user units.

For both selection layer types (marker and mask), the resulting areas of interest created by different polygons can overlap. Each area of interest region is processed separately; however, the result is always merged. In some cases, the area of interest is increased in size depending on the transformation parameters.

When the *selection_layer* argument is specified, the applied transformation only produces results inside the specified areas of interest. For some transformations, this requires the operation to consider input geometries that are outside of the areas of interest.

If the *selection_layer* argument is *not* specified, the transformation is applied to the entire input layer.

- *line_end_layer*‘(‘**LINE_END**‘)’

An optional argument specifying a layer that defines a line-end shape. This layer must only define a line-end shape as specified by **LINE_END** in “[DFM Spec Reshape](#)” on page 299. It is only compatible with specifications for **CORNER** and **LINE_END** transformations. The *line_end_layer* can be one of two types:

Polygon — If *line_end_layer* is a polygon layer, edges of the input layer that interact with a line-end polygon are treated as a line-end edge.

Edge — If *line_end_layer* is an edge layer, the edges of the input layer that touch line-end edges are treated as a line-end edge.

The following behavior applies:

- If *line_end_layer* is specified, process line-end correction happens only on the line-end edges.
- If *line_end_layer* is specified with **CORNER** transformation, **CONCAVE_SPEC** and **CONVEX_SPEC** values do not apply on the line-end edges.

Description

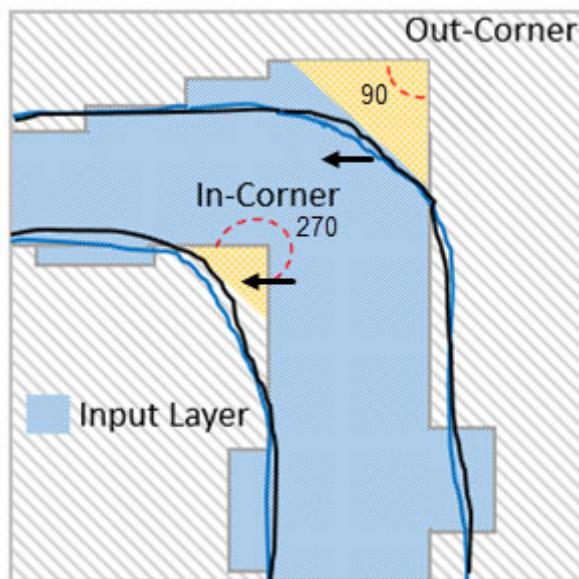
The DFM Reshape operation transforms optical lithography shapes from the input layer (typically, post metal biasing and retargeting) for a subsequent [DFM Via Shift](#) operation. The DFM Reshape operation generates shapes that imitate the actual metal shapes to improve the metal enclosure of vias in corner and line-end locations and to output uniform via shapes from the input layer. Resulting metal shapes from the operation are on the specified output layer. The

correction values for the corner and line-end shapes are specified by a [DFM Spec Reshape](#) specification, which is referenced by the DFM Reshape operation.

For corners, the metal shapes are defined by out-corner and in-corner geometry transformations. These shape transformations are used to imitate the actual metal shape and guide the via location away from the out-corner and towards the in-corner. Shape transformations are generated as isosceles triangles, sized according to the length of the corner-pair edge, with longer edges resulting in larger triangles. See [Figure 11-14](#).

- **Out-Corner** — The shape transformations are corner-chop triangles with legs that are coincident convex edge pairs. The definition of the convex edge pair, the out-corner, is two adjacent edges having angle == 90. A NOT operation is performed on the retarget metal layer. This results in a corner-chop region that guides via movement in a subsequent DFM Via Shift operation away from this region, potentially improving metal coverage.
- **In-Corner** — The shape transformations are corner-fill triangles with legs that are coincident concave edge pairs. The definition of the concave edge pair, the in-corner, is two adjacent edges having angle == 270. An OR operation is performed on the retarget metal layer. This results in a corner-fill region that enables via movement in a subsequent DFM Via Shift operation towards this region, potentially improving metal coverage.

Figure 11-14. Out-Corner and In-Corner Shape Transformations



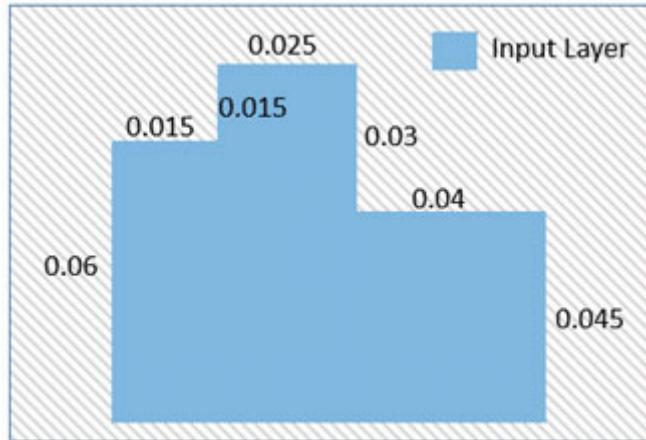
For line-ends, the metal shapes are defined by trim-rectangle and extension-rectangle shape transformations. The resulting rectangles correct the input metal line by making it shorter (trim-rectangle) or longer (extension-rectangle). These line-end transformations enable via movement in a subsequent DFM Via Shift operation away from the trim-rectangle region and towards the extension-rectangle region to improve via enclosure. Rectangles are sized according to length (perpendicular to the line-end edge) and width (parallel to the line-end edge) values as

specified by the DFM Spec Reshape LINE-END specification. See “[Example 3 — DFM RESHAPE LINE-END Specifications](#)” on page 314.

Examples

Example 1 — DFM RESHAPE CORNER Specifications

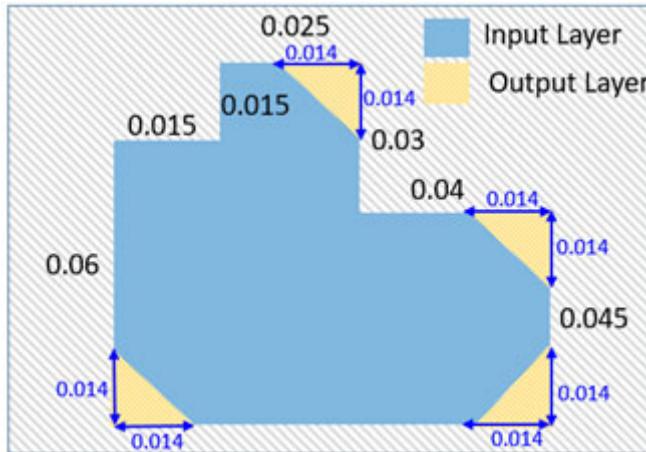
Figure 11-15. CORNER (Input)



Case 1: The following example generates corner-chop triangles at convex corners with length 0.014 when both edges are ≥ 0.022 .

```
DFM SPEC RESHAPE corner_chop_spec CORNER 0.014(0.022)
low_metal_chop = DFM RESHAPE CORNER_CONVEX(CONVEX_SPEC corner_chop_spec)
m_low
```

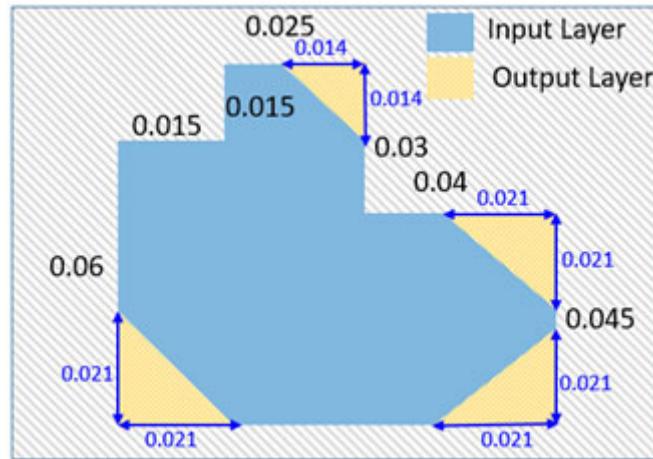
Figure 11-16. Case 1: CORNER (CORNER_CONVEX)



Case 2: The following example generates corner-chop triangles at convex corners with length 0.014 when both edges are ≥ 0.022 . It generates corner-chop triangles with length 0.021 at convex corners when both edges ≥ 0.033 .

```
DFM SPEC RESHAPE corner_chop_spec CORNER 0.014(0.022) 0.021(0.033)
low_metal_chop = DFM RESHAPE CORNER_CONVEX(CONVEX_SPEC corner_chop_spec)
m_low
```

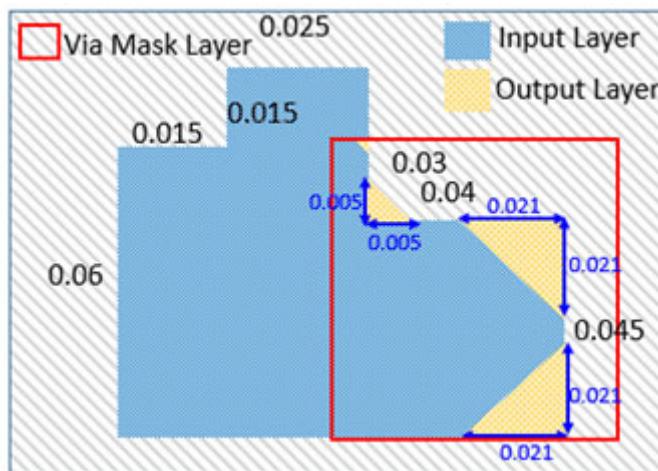
Figure 11-17. Case 2: CORNER (CORNER_CONVEX Edge Pairs)



Case 3: The following example concurrently generates corner-chop triangles at convex corners and corner-fill triangles at concave corners, inside a sized via mask layer that is a selection layer. The output layer is clipped at the boundary of the selection layer.

```
DFM SPEC RESHAPE corner_chop_spec CORNER 0.014(0.022) 0.021(0.033)
DFM SPEC RESHAPE corner_fill_spec CORNER 0.005(0.022) 0.015(0.035)
low_metal_chop = DFM RESHAPE CORNER_CONVEX(CONVEX_SPEC corner_chop_spec
    CONCAVE_SPEC corner_fill_spec) m_low via(MASK 0.02)
low_metal_fill = DFM RESHAPE CORNER_CONCAVE(CONVEX_SPEC corner_chop_spec
    CONCAVE_SPEC corner_fill_spec) m_low via(MASK 0.02)
```

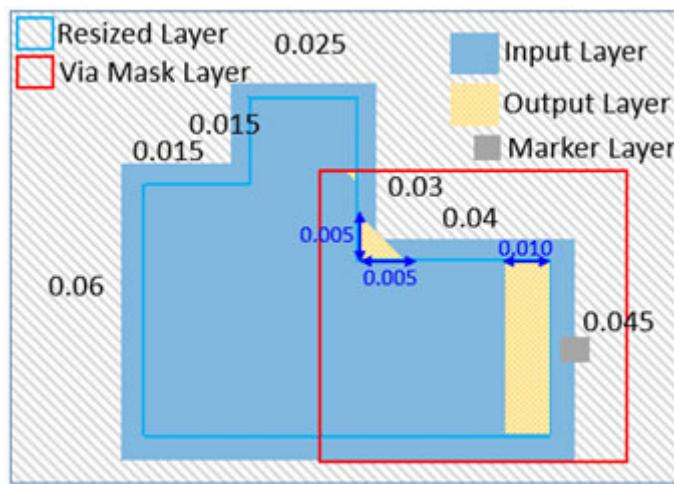
Figure 11-18. Case 3: CORNER (CORNER_CONVEX and CORNER_CONCAVE with MASK)



Case 4: The following example concurrently generates corner-chop and corner-fill triangles and performs line-end correction on a resized input layer, inside a sized via mask layer that is a selection layer. The output layer is clipped at the boundary of the selection layer.

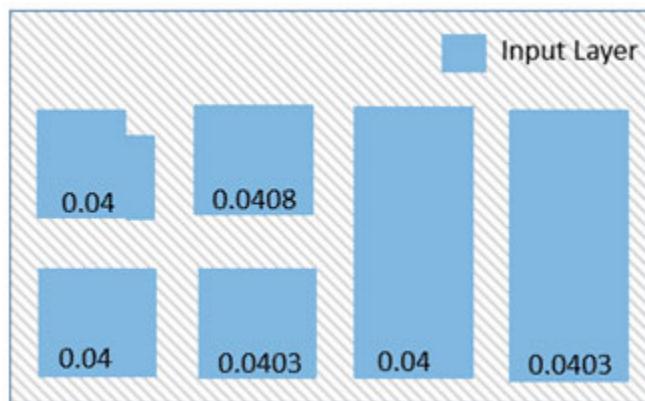
```
DFM SPEC RESHAPE corner_chop_spec CORNER 0.014(0.022) 0.021(0.033)
DFM SPEC RESHAPE corner_fill_spec CORNER 0.005(0.022) 0.015(0.035)
DFM SPEC RESHAPE line_end_spec LINE_END -0.01
resized_low_metal_chop = DFM RESHAPE CORNER_CONVEX(CONVEX_SPEC
    corner_chop_spec CONCAVE_SPEC corner_fill_spec LINE_END_SPEC
    line_end_spec) m_low(RESIZE (-0.003)) via(MASK 0.02)
    line_end_marker(LINE_END)
resized_low_metal_fill = DFM RESHAPE CORNER_CONCAVE(CONVEX_SPEC
    corner_chop_spec CONCAVE_SPEC corner_fill_spec LINE_END_SPEC
    line_end_spec) m_low(RESIZE (-0.003)) via(MASK 0.02)
    line_end_marker(LINE_END)
```

Figure 11-19. Case 4: CORNER and LINE_END with MASK and RESIZE Attribute



Example 2 — DFM RESHAPE UNIFORM Specifications

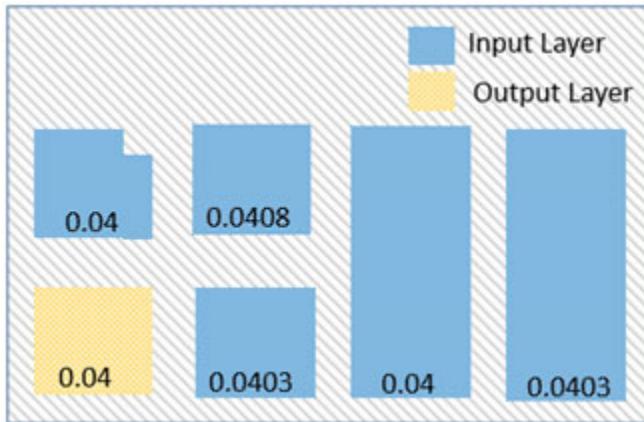
Figure 11-20. UNIFORM (Input)



Case 1: The following example selects uniform square vias of size 0.04.

```
square_via = DFM RESHAPE UNIFORM via(BY RECTANGLE 0.040)
```

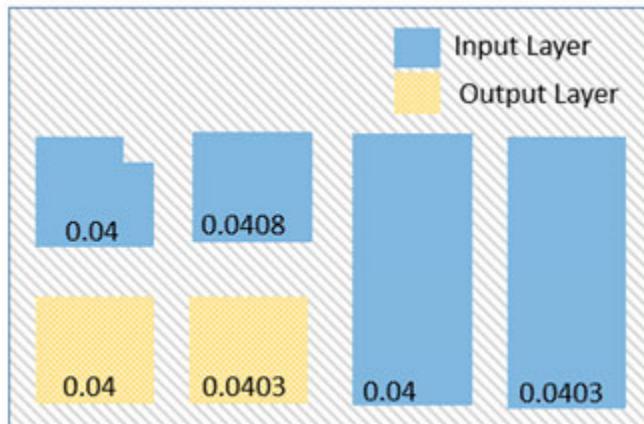
Figure 11-21. Case 1: UNIFORM BY RECTANGLE (Square)



Case 2: The following example selects square vias of size 0.04 with a tolerance of 0.0005. The output is uniform square vias of size 0.04, which are snapped to the specified grid. Although not visible in the figure, the input via of size 0.0403 is resized to a 0.040 square on output.

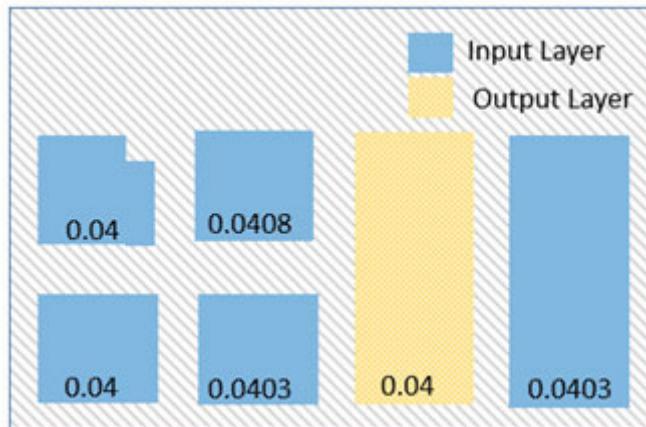
```
square_via_reshape = DFM RESHAPE UNIFORM via(BY RECTANGLE 0.040 TOLERANCE  
0.0005 GRID 4000)
```

**Figure 11-22. Case 2: UNIFORM By RECTANGLE (Square with TOLERANCE
and GRID)**



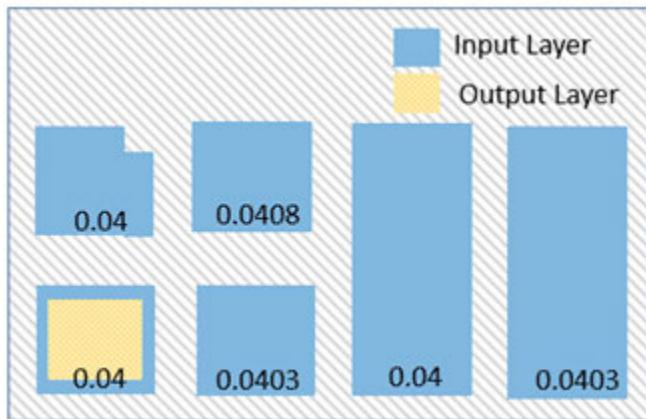
Case 3: The following example selects uniform rectangular vias of size 0.1 x 0.04, and then snaps the output on the specified grid.

```
rect_via_reshape = DFM RESHAPE UNIFORM via(BY RECTANGLE 0.1 0.040 GRID  
4000)
```

Figure 11-23. Case 3: UNIFORM BY RECTANGLE (Rectangle with GRID)

Case 4: The following example selects uniform square vias and resizes them from 0.04 to 0.03.

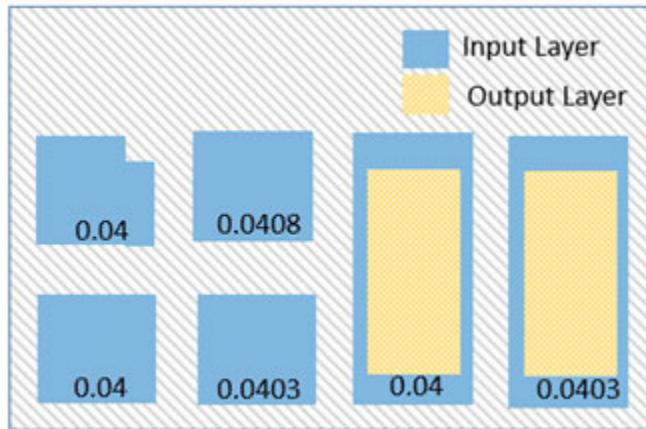
```
square_resized_via = DFM RESHAPE UNIFORM via(BY RECTANGLE  
0.040::0.030)
```

Figure 11-24. Case 4: UNIFORM BY RECTANGLE (Square Resized)

Case 5: The following example selects uniform rectangular vias with tolerance 0.0005 and resizes them from 0.1 x 0.04 to 0.07 x 0.03, and then snaps the output on the specified grid.

```
rect_resized_via_reshape = DFM RESHAPE UNIFORM via(BY RECTANGLE  
0.1::0.07 0.040::0.03 TOLERANCE 0.0005 GRID 4000)
```

Figure 11-25. Case 5: UNIFORM BY RECTANGLE (Rectangle Resized with TOLERANCE and GRID)

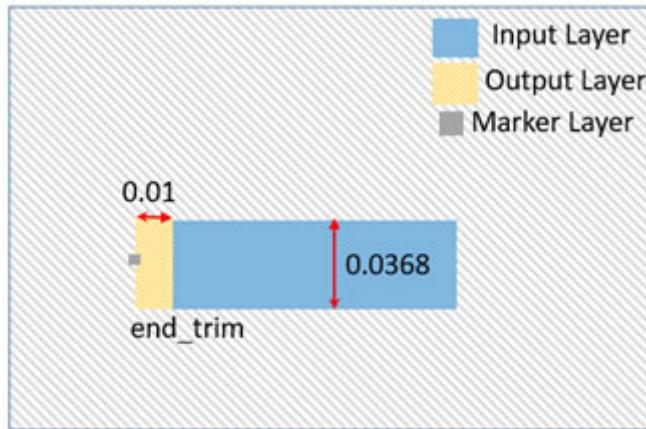


Example 3 — DFM RESHAPE LINE-END Specifications

Case 1: The following example performs line-end correction by generating a trim-rectangle inward from the line-end edge by -0.01 when the line-end width is ≥ 0.20 . The DFM SPEC RESHAPE spec_name is referenced by the DFM RESHAPE LINE-END_PULLBACK statement.

```
DFM SPEC RESHAPE line_end_spec LINE_END -0.010(0.020) 0.015(0.050)
end_trim = DFM RESHAPE LINE-END_PULLBACK(LINE-END_SPEC line_end_spec)
metal_in via_reshape(MASK 0.05) end_marker_in(LINE-END)
```

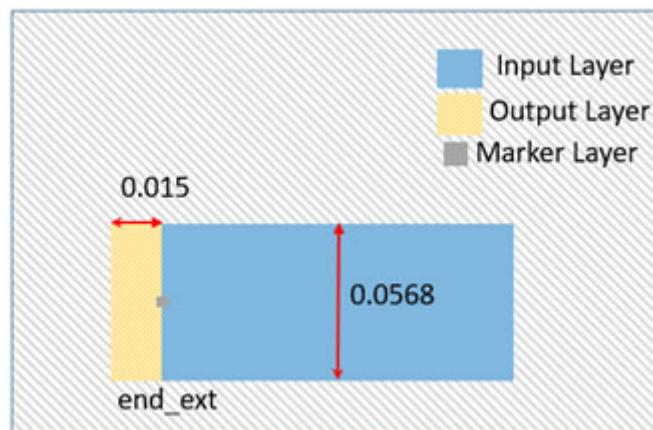
Figure 11-26. Case 1: DFM RESHAPE LINE-END_PULLBACK



Case 2: The following example performs line-end correction by generating an extension-rectangle outward from the line-end edge by 0.15 when the width is ≥ 0.05 . The DFM SPEC RESHAPE spec_name is referenced by the DFM RESHAPE LINE-END_EXTENSION statement.

```
DFM SPEC RESHAPE line_end_spec LINE_END -0.010(0.020) 0.015(0.050)
end_ext = DFM RESHAPE LINE-END_EXTENSION(LINE-END_SPEC line_end_spec)
metal_in via_reshape(MASK 0.05) end_marker_in(LINE-END)
```

Figure 11-27. Case 2: DFM RESHAPE LINE_END_EXTENSION



DFM RDB Reference

The Calibre YieldAnalyzer and Calibre YieldEnhancer toolsets use proprietary DFM Results Databases (RDBs) to store the types of data required for analysis and enhancement.

These include geometric data, connectivity information, object properties and statistics.

The basic contents of all Calibre RDBs are the same, though the specifics vary according to the types of information they contain and the operation and method used to create the RDBs.

Table 11-1. DFM RDBs, Summary

RDB	Generated by...	Contains	Node Aware?
Scores RBD	DFM Analyze	Scores	N/A
Check Results RDB	DFM Measure	Scores, Statistics	N/A
Transitions RDB	DFM Transition DFM RDB	Transitions	Yes
Single-Layer RDB	DFM RDB	Geometries	Yes
Multilayer RDB	DFM RDB	Geometries	Yes
Fill RDB	DFM Fill	Fill Geometries	N/A

Making maximum use of DFM RDBs may involve reformatting or manipulating the data for use in other applications. Because they are ASCII files, these RDBs are easily manipulated by various scripting languages. This reference material is designed to aid you in working with RDB files directly by providing detailed descriptions of the files generated by each of the DFM operations.

Scores RDB File Format

Output from: [DFM Analyze](#)

Contains scores or statistics generated by DFM Analyze.

RDB Organization

The RDB is organized as follows, depending on how the analysis is performed:

- WINDOW Analysis — One result data block per capture window.
The default behavior (without WINDOW or BY CELL specified) is WINDOW analysis with one capture window consisting of the database extent, resulting in one result data block.
- BY CELL Analysis — One result data block per cell

Scores and statistics are reported in the results properties associated with each result data block. The result properties that are reported depend on the expression evaluated in the DFM Analyze operation and the keywords that are specified.

Format

The results database (RDB) is an ASCII file with the following contents:

```
header
results_name
results_info
check_text
result_data
result_data
...
...
```

There is one result_data block for each cell or window in the analysis. The result_data block has the following format:

```
result_type
result_properties
result_coordinates
```

The following image shows a Scores RDB with annotations. The example RDB is cell-based but the annotations explain the content for both a cell-based and window-based RDB. A cell-based RDB has a result_data block for each cell and includes the DN property with the cell name. A window-based RDB has a result_data block for each capture window and does not include the DN property.

Figure 11-28. Scores RDB from DFM Analyze

```

header      TOP 1000      top cell and precision
results_name k:::<1>
results_info 14621 14621 2 Jul 21 10:03:17 2003
check_text   k:::<1> = DFM ANALYZE POLY GATE_EDGE GATE_WIDTH
(layer derivation)   WINDOW 105 [AREA(POLY)+PERIMETER(POLY)+LENGTH(GATE_EDGE)+COUNT(GATE_WIDTH)+AREA() +PERIMETER())]
p 1 4
DN SENSEAMP
DV 24572.25 (325677545.425)
DA 11025 (6.54E24)
DA POLY 5462.85 (235439E13)
DP 420 (5790)
DP POLY 134.6 (3253476E34)
DL GATE_EDGE 346.8 (346543.56E13)
DC GATE_WIDTH 23 (456786)
-8986400 -4272200
-8881400 -4272200
-8881400 -4167200
-8986400 -4167200
p 2 4      result_type (type, ordinal, # vertices)
DN RAMBIT
DV 1450.25 (45462.85)
DA 2105 (6.54E24)
DA POLY 2462.85 (45339E13)
DP 584 (5790)
DP POLY 144.6 (32571476E54)
DL GATE_EDGE 346.8 (3544.56E13)
DC GATE_WIDTH 23 (4258)
-8986400 -3852200
-8881400 -3852200
-8881400 -3747200
-8986400 -3747200
:
:
result_data (for first cell or capture window)
result_data (for second cell or capture window)
additional result_data blocks for each cell or capture window
result_properties (flat value in parentheses for BY CELL analysis)
result_coordinates (cell extent or capture window)

```

Parameters

- *header*
The name of the top cell and the RDB precision. Example: top 1000
- *results_name*
A string representing the name for the results (analogous to the rule check name for a DRC results database). The string has the following format, depending on whether the DFM Analyze operation is within a rule check:

Within a Rule Check	Not Within a Rule Check
<i>rulecheck::<n></i>	<i>layer</i>

rulecheck — The rule check name.

n — The ordinal within the rule check of the DFM Analyze operation that generates the RDB. The ordinal is enclosed in angle brackets (<*n*>).

layer — The name of the derived layer created by the DFM Analyze operation.

- *results_info*

```
flat_count hier_count num_text month day time year
```

One line containing the flat result count, the hierarchical result count, the number of check text lines that follow, and a date stamp (month day time year).

For example: 9 9 3 Nov 17 15:43:02 2016

- *check_text*

One or more lines, where the first line contains the DFM Analyze layer derivation. If the DFM Analyze operation is within a rule check, layers derived within the rule check are prefixed with “*rulecheck::*”.

Additional lines are included if the DFM Analyze operation includes the COMMENT keyword with a comment string. Each COMMENT keyword set adds one line of check text.

- *result_data*

There is one *result_data* block for each capture window or cell. The *result_data* block has the following format:

```
result_type
result_properties
result_coordinates
```

result_type — One line with the format:

```
type ordinal num_objects
```

type — The result type, indicated by the character “p” for polygon.

ordinal — The ordinal of the capture geometry within the RDB. This reflects the order in which the analysis was performed.

num_objects — The number of vertices that make up the polygon, which is always four, indicating a rectangular capture geometry. The capture geometry corresponds to the cell extent or capture window.

For example, “p 2 4” indicates a polygon which is the second object within the RDB and has four vertices.

result_properties

The result properties that are reported depend on the expression evaluated in the DFM Analyze operation and the keywords used. The keywords DV and DX control the properties reported; both keywords are included by default.

Each line contains the property name followed by its value. When the analysis is performed BY CELL, the numeric properties report two values: the cell level value

and the flat value (as if the cell were flattened in all of its placements). The flat value is shown in parentheses.

Table 11-2. Scores RDB Result Properties for DFM Analyze

Property	Description
DV	Value of the expression.
DA [<i>layer</i>]	Total area. Included if AREA() or AREA(<i>layer</i>) appears in the expression. <ul style="list-style-type: none"> • AREA() — Area of the capture geometry. • AREA(<i>layer</i>) — Total area of <i>layer</i> in the capture geometry.
DP [<i>layer</i>]	Total perimeter. Included if PERIMETER() or PERIMETER(<i>layer</i>) appears in the expression. <ul style="list-style-type: none"> • PERIMETER() — Perimeter of the capture geometry. • PERIMETER(<i>layer</i>) — Total perimeter of <i>layer</i> polygons in the capture geometry.
DL <i>layer</i>	Total length of <i>layer</i> edges in the capture geometry. Included if LENGTH(<i>layer</i>) appears in the expression.
DC <i>layer</i>	Total count of <i>layer</i> objects in the capture geometry. Included if COUNT(<i>layer</i>) appears in the expression.
DN	Cell name. Included if the RDB is created using the BY CELL keyword in DFM Analyze.
DEC <i>layer</i>	Total edge projection length—the sum of projection lengths for all the error clusters on the <i>layer</i> and within the capture geometry. Included if EC(<i>layer</i>) appears in the expression.
DEW <i>layer</i>	Total edge separation—sum of the edge separations for all the error clusters on the <i>layer</i> and within the capture geometry. Included if EW(<i>layer</i>) appears in the expression.

result_coordinates — The coordinates of the capture geometry. There is one line for each vertex, where each line contains one pair of coordinates. The result coordinates give the four vertices of the capture geometry (cell extent or capture window).

Result coordinates are controlled by the COORD keyword, which is included by default.

Check Results RDB File Format

Output from: [DFM Measure](#)

Contains an ASCII representation of the geometric results from a DFM Measure operation and statistics about the results.

RDB Organization

Results are organized by rule check, where there is one rule check for each analysis region per measurement rule. The analysis region is determined by the WINDOW or BY CELL keyword in the RDB specification. Each rule check contains one check result for each output geometry that *intersects* the analysis region, where the meaning of intersect depends on the type of output, as described in the following table.

Output Type	A result <i>intersects</i> the analysis region if ...
Error-directed	The edge cluster has an extent that shares area with the window or cell.
Polygon	The polygon shares area with the window or cell, as determined with a Boolean AND.
Edge	The edge shares points with the window or cell (including the window or cell periphery), as determined with the Not Outside Edge operation.

All geometric results and statistics are reported as if the operation were executed completely flat in the analysis region. Statistics calculated by DFM Measure are reported in the rulecheck text.

Format

The results database (RDB) is an ASCII file with the following contents:

```
header
rule_check_results
rule_check_results
...

```

The rule_check_results block has the following format, where the contents depend on the keywords used in the DFM Measure operation.

```
rulecheck_name
rulecheck_info
check_text
result_data
result_data
...

```

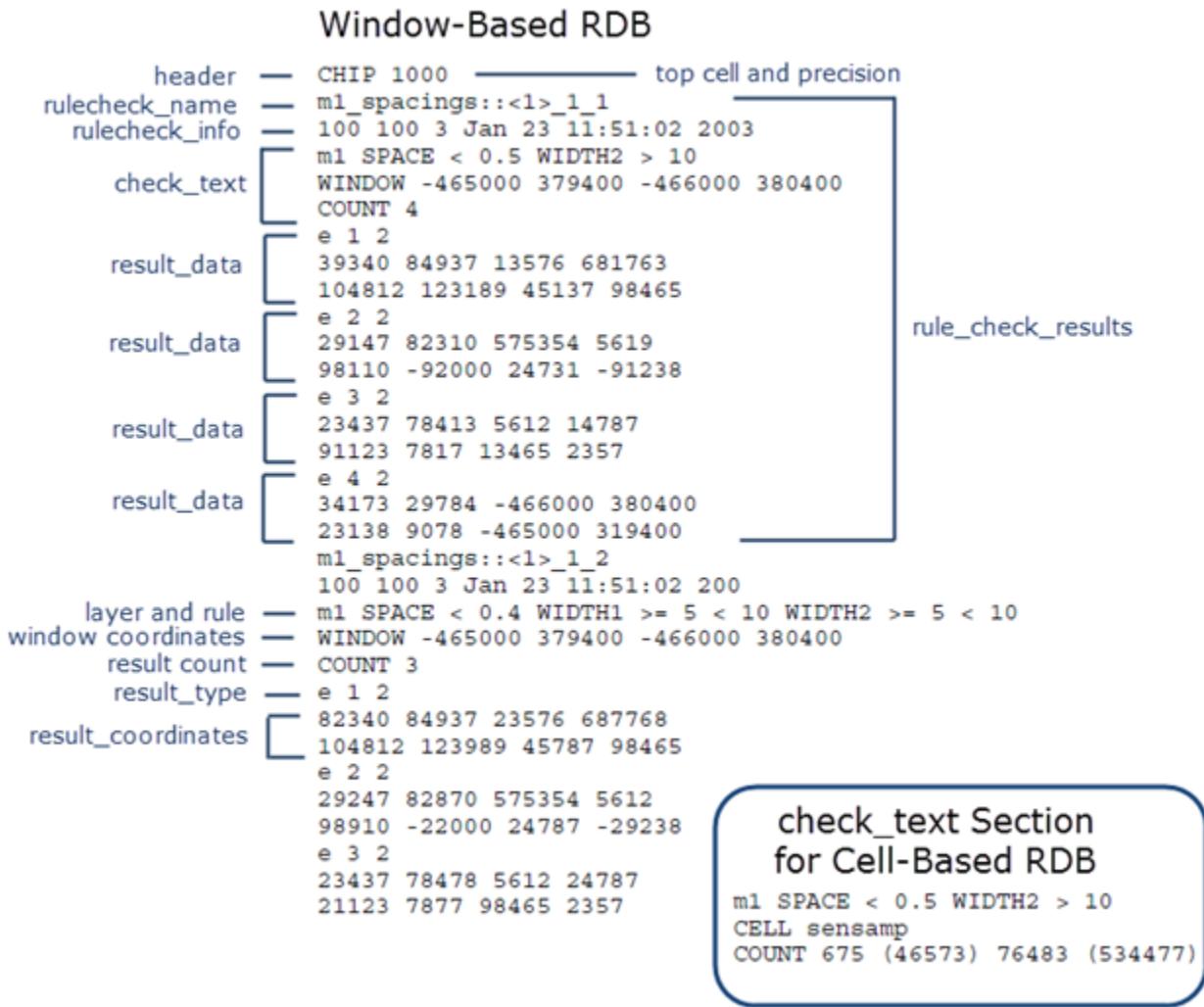
The result_data block has the following format:

```
result_type
result_coordinates

```

The following image shows a window-based Check Results RDB. The check_text block for a cell-based RDB is shown in an inset. The output type is error-directed.

Figure 11-29. Check Results RDB File Format



Parameters

- *header*

The name of the top cell and the RDB precision. Example: top 1000

- *rule_check_results*

The *rule_check_results* block includes all results for a particular rule check. The structure of the RDB depends on the WINDOW and BY CELL keywords in the RDB specification:

WINDOW — One *rule_check_results* block is written per measurement per window.

BY CELLS — One *rule_check_results* block is written per measurement per cell.

- *rulecheck_name*

A string representing the rule check name, which has the following format:

name_measurement_analysis

name — The *name* depends on whether the DFM Measure operation is within a rule check:

In a rule check — *name* has the form *checkname*::<*n*>, where *checkname* is the name of the rule check and <*n*> indicates that this is the *n*th layer generated during the rule check.

Not in a rule check — *name* is the name of the derived layer created by the DFM Measure operation.

measurement — The ordinal, starting from 1, of the measurement rule within the DFM Measure operation.

analysis — The ordinal, starting from 1, of the analysis region in the evaluation sequence.

- *rulecheck_info*

flat_count hier_count num_text month day time year

One line containing the flat result count, the hierarchical result count, the number of check text lines that follow, and a date stamp (month day time year).

For example: 9 9 3 Nov 17 15:43:02 2016

- *check_text*

Three lines or four lines of text, where the content depends on the use of the WINDOW or BY CELL keyword in the RDB specification and the type of output (error-directed, polygon or edge).

```
layer_and_rule
{ CELL cellname | WINDOW x_ll y_ll x_ur y_ur }
COUNT count [(count_f) count_e (count_ef)]
[ AREA data | LENGTH data ]
```

layer_and_rule — One line containing the layer name and the measurement rule.

CELL *cellname* — One line with the cell name. This line is included if BY CELL is specified.

WINDOW *x_ll y_ll x_ur y_ur* — One line with the coordinates of the lower-left and upper-right corners of the analysis window, in database units. This line is included if WINDOW is specified.

COUNT *count* [(*count_f*) *count_e* (*count_ef*)] — One line providing the results count(s) as follows:

- WINDOW analysis — One result count is provided, where *count* is the flat count of results that intersect the window.

- BY CELL analysis — Four results counts are provided, giving the count of results that intersect the cell. The following result counts are given:
 - *count* — count in the immediate hierarchical level from the perspective of the operation
 - *(count_f)* — total count in all flattened instantiations of the cell
 - *count_e* — count with the cell subhierarchy expanded
 - *(count_ef)* — total count in all expanded, flat instantiations of the cell

See “[RDB Organization](#)” on page 321 for the definition of intersect depending on the result type.

AREA *data* — An optional line that is included if the output is polygon data. The *data* value is the total area of all output polygons that *intersect* the window or cell, as calculated with a Boolean AND operation.

LENGTH *data* — An optional line that is included if the output is edge data. The *data* value is the total length of all output edges that share points in common with the analysis window or cell, as calculated with a Not Outside Edge operation.

- ***result_data***

There is a *result_data* block for each result that intersects the cell or window; see “[RDB Organization](#)” on page 321. The *result_data* block has the following format:

```
result_type
result_coordinates
```

result_type — One line with the format:

```
type ordinal num_objects
```

type — The result type, indicated by the character “p” for a polygon or “e” for an edge or error cluster.

ordinal — The ordinal of the geometry within the rule check.

num_objects — The number of objects that make up the result. For polygons, this is number of vertices, for edge and error clusters, it is the number of edges.

For example, “p 2 4” indicates a polygon which is the second object within the rule check and has four vertices.

result_coordinates — The coordinates for the geometry. Coordinates are supplied in top cell space using an arbitrarily selected placement of the cell.

polygon data — One line for each vertex. Each line contains one pair of coordinates.

error and edge data — One line for each edge. Each line contains two pairs of coordinates giving the end points of the edge.

Transitions RDB File Format

Output from: A [DFM Transition](#) operation with an RDB specification or a [DFM RDB](#) operation with the TRANSITION keyword.

Contains an ASCII representation of transition geometries and statistics. The transition geometries are on one via layer and two interconnect layers.

There is one rule check for each capture window or cell. Each rule check reports statistics regarding the total number of transitions that intersect the window or cell—these statistics are reported as check text. A rule check also includes one result for each transition geometry that was created. All geometric results and statistics are reported as if the operation were executed completely flat.

Format

The results database (RDB) is an ASCII file with the following contents:

```
header
rule_check_results
rule_check_results
...
...
```

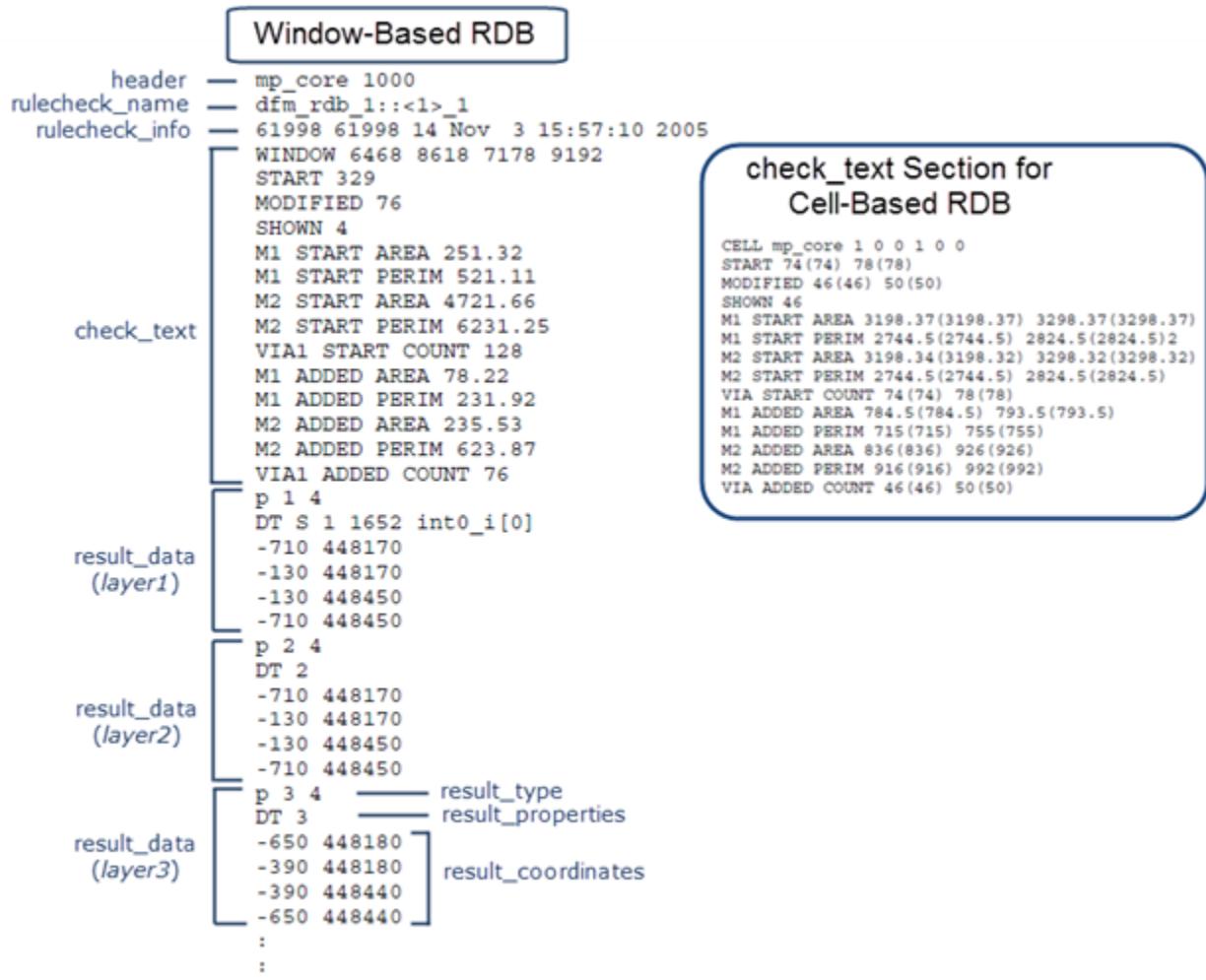
The rule_check_results block has the following format, where the contents depend on the keywords used in the operation that creates the RDB.

```
rulecheck_name
rulecheck_info
check_text
result_data
result_data
...
...
```

The result_data block has the following format:

```
result_type
result_properties
result_coordinates
```

Figure 11-30. Transitions RDB File Format



Parameters

- **header**
The name of the top cell and the RDB precision. Example: top 1000
- **rule_check_results**
There is one *rule_check_results* block for each capture window or cell.
- **rulecheck_name**
A string representing the rule check name. The *rulecheck_name* depends on the conditions in which the RDB is created. The general form is the following:

Table 11-3. *rulecheck_name* Format in Transitions RDB

RDB Creation Condition	<i>rulecheck_name</i> format
Within a rule check	<i>scope</i> ::< <i>n</i> >[_ <i>m</i>]

Table 11-3. rulecheck_name Format in Transitions RDB (cont.)

RDB Creation Condition	<i>rulecheck_name</i> format
DFM RDB operation with CHECKNAME keyword	[<i>scope</i> ::] <i>name</i> [<i>_m</i>]
Layer derivation outside of a rule check	<i>derived_layer_name</i>

scope — The name of the rule check that contains the DFM Transition or DFM RDB operation.

When CHECKNAME is specified with a DFM RDB operation, *scope* is only included if “CHECKNAME “%_l”” is specified and the first input layer is derived within the rule check (a local layer).

derived_layer_name — The name of the derived layer created by the DFM Transition operation.

n — The ordinal within the rule check of the DFM RDB operation that writes out the layer data to the RDB. The ordinal is enclosed in angle brackets (<*n*>).

m — A positive integer that uniquely identifies the window or cell.

name — The *name* parameter specified with the CHECKNAME keyword in a DFM RDB operation. If pattern substitution is used, *name* is replaced with the name of the first input layer name for %_l_, and replaced with the top cell name for %_t_.

For examples of *rulecheck_name* construction for the DFM RDB operation, see “Single Layer RDB File Format.”

- *rulecheck_info*

```
flat_count hier_count num_text month day time year
```

One line containing the flat result count, the hierarchical result count, the number of check text lines that follow, and a date stamp (month day time year).

For example: 9 9 1 Nov 17 15:43:02 2016

- *check_text*

Fourteen lines of text, where the content depends on whether the reporting is by cell or by window. If the RDB was created with a DFM RDB operation, only the CELL and SHOWN lines have meaningful data—all other check text lines have 0 for the value.

The check text has the following format:

```
{WINDOW x_ll y_ll x_ur y_ur | CELL cell_name cell_transform}
START num [(num_f) num_e (num_ef)]
MODIFIED num [(num_f) num_e (num_ef)]
SHOWN num
layerA START AREA area [(area_f) area_e (area_ef)]
layerA START PERIM perim [(perim_f) perim_e (perim_ef)]
layerB START AREA area [(area_f) area_e (area_ef)]
layerB START PERIM perim [(perim_f) perim_e (perim_ef)]
layerC START COUNT count [(count_f) count_e (count_ef)]
layerA ADDED AREA area [(area_f) area_e (area_ef)]
layerA ADDED PERIM perim [(perim_f) perim_e (perim_ef)]
layerB ADDED AREA area [(area_f) area_e (area_ef)]
layerB ADDED PERIM perim [(perim_f) perim_e (perim_ef)]
layerC ADDED COUNT count [(count_f) count_e (count_ef)]
```

WINDOW *x_ll y_ll x_ur y_ur* — Used with window analysis in a DFM Transitions operation. The four values give the coordinates of the lower-left and upper-right corners of the data capture window, in database units.

CELL *cell_name cell_transform* — Used when the BY CELL keyword is included in the DFM Transition operation or a DFM RDB operation is used to create the RDB. The values give the name of the cell and six numbers defining a transform matrix that maps global coordinates to the coordinates of the cell. See DRC Cell Name for a description of the transform matrix.

With the exception of the SHOWN check text line, the content of the remaining check text lines depends on whether the analysis was WINDOW or BY CELL:

- **WINDOW** analysis — One data value
- **BY CELL** analysis — Four data values:
 - data* — for the immediate hierarchical level
 - (data_f)* — for the flattened instantiations of the cell
 - data_e* — for the cell subhierarchy expanded
 - (data_ef)* — for all expanded, flat instantiations of the cell

START *num* — The number of transitions (not polygons) that intersect the analysis window or cell.

MODIFIED *num* — The number of modified transitions (not polygons) that intersect the analysis window or cell.

SHOWN *num* — The count of all modified transitions (not polygons) actually shown in the RDB. This number is controlled by the MAXIMUM keyword.

layerA*, *layerB*, and *layerC — These indicate the layer that the data applies to. *layerA* and *layerB* are the first and second interconnect layers in a DFM Transition operation, or the second and third layers in a DFM RDB operation. *layerC* is the via layer in a DFM Transition operation or the first layer in a DFM RDB operation.

START [AREA | PERIM | COUNT] — Gives the total area, total perimeter, or via count, respectively, before transition modification.

ADDED [AREA | PERIM | COUNT] — Gives the total area, total perimeter, or via count, respectively, after transition modification.

- *result_data*

There is a *result_data* block for each transition geometry, with the following format:

```
result_type
result_properties
result_coordinates
```

result_type — One line with the format:

```
type ordinal num_objects
```

type — The result type, indicated by the character “p” for a polygon.

ordinal — The ordinal of the geometry within the rule check.

num_objects — The number of vertices in the polygon.

For example, “p 2 4” indicates a polygon which is the second object within the rule check and has four vertices.

result_properties — One line starting with “DT” and with the format:

```
DT [S] layer_num [net_number [net_name]]
```

S — Indicates the polygon is the start of a modified transition.

layer_num — The numeral 1, 2, or 3, indicating which input layer the result is from.

net_number [net_name] — Nodal information is included if the NODAL keyword is specified. The internal node number is given, along with the net name if available. This connectivity is associated only with the polygon that is the start of the transition.

The node number is a property of the improved transition itself, not of the individual geometries in the improved transition. The node number of an improved transition is that of the original *layer3* shape in the transition. In the unlikely event that a START constraint other than == 1 is specified, and the *layer3* geometry in the transition collectively has more than one node number, then a node number is chosen arbitrarily.

result_coordinates — The coordinates for the geometry. Coordinates are supplied in top cell space using an arbitrarily selected placement of the cell. There is one line for each vertex, and each line contains one pair of coordinates.

Single Layer RDB File Format

Output from: A one-layer DFM RDB operation.

Contains an ASCII representation of one original or derived layer.

Note

 RDBs can also be generated by layer operations that include the RDB keyword, such as Density and Net Area Ratio. In such cases the RDB format is described with the command reference.

Format

The results database (RDB) is an ASCII file with the following contents:

```
header
rule_check_results
rule_check_results
...
...
```

The rule_check_results block has the following format, where the contents depend on the keywords used in the DFM RDB operation.

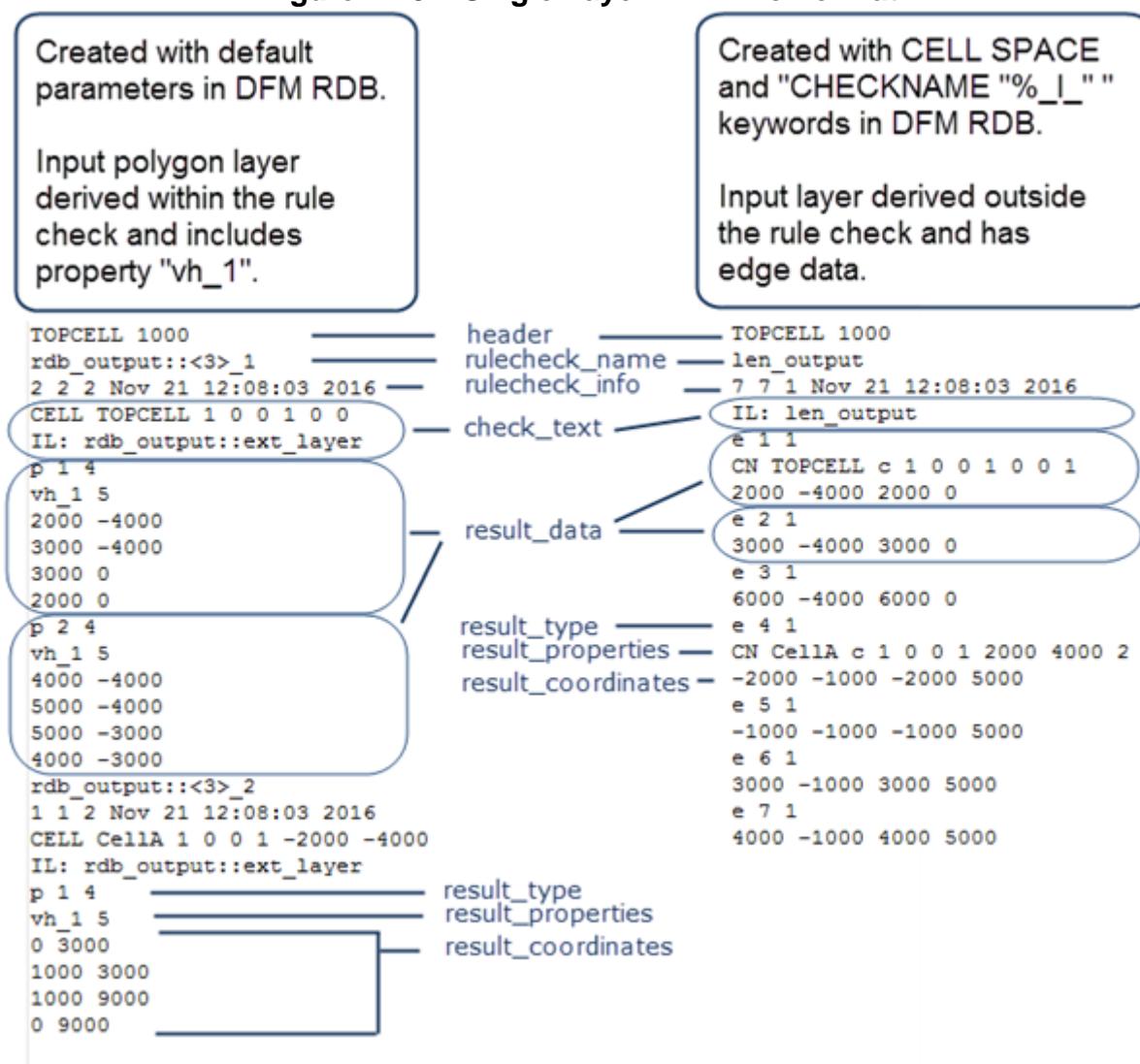
```
rulecheck_name
rulecheck_info
check_text
result_data
result_data
...
...
```

The result_data block has the following format:

```
result_type
result_properties
result_coordinates
```

The following image displays two different single layer RDBs with annotations pointing out the different elements in the file structure. The RDB on the left was created with default parameters. The input layer was a polygon layer derived within the rule check, and it includes the property “vh_1”. The RDB on the right was created with the CELL SPACE and “CHECKNAME “%_1_”” keywords. The input layer was derived outside the rule check and contains edge data. The different elements in the file format are described in the “Parameters” section following the figure.

Figure 11-31. Single Layer RDB File Format



Parameters

- *header*

The name of the top cell and the RDB precision. Example: top 1000

- *rule_check_results*

The *rule_check_results* block includes all results for a particular rule check. The block contains the rule check name, check information, rule check text, and result data. The number of *rule_check_results* blocks is determined by the ALL CELLS keyword in the DFM RDB operation:

Default — One *rule_check_results* block is written for each cell in the layout. The *check_text* block includes the CELL check text line, unless the CELL SPACE keyword is used, in which case the first geometry in each cell is annotated with the CN result property; see the *result_properties* definition.

ALL CELLS — A single *rulecheck_results* block is written. The first geometry in each cell is annotated with the CN result property.

- *rulecheck_name*

A string representing the rule check name. The *rulecheck_name* depends on the keywords used in the DFM RDB operation and whether the input layer was derived within the scope of the rule check containing the DFM RDB operation. The general form is the following, depending on whether CHECKNAME is used:

Table 11-4. rulecheck_name Format in Single Layer RDB

Without CHECKNAME	With CHECKNAME
<i>scope::<n>[_m]</i>	[<i>scope::</i>] <i>name[_m]</i>

scope — The name of the rule check that contains the DFM RDB operation. When CHECKNAME is specified, *scope* is only included if “CHECKNAME “%_l” is specified and the input layer is derived within the rule check (a local layer).

n — The ordinal within the rule check of the DFM RDB operation that writes out the layer data to the RDB. The ordinal is enclosed in angle brackets (<*n*>).

m — A positive integer that uniquely identifies the cell. This integer is not used if the ALL CELLS or CELL SPACE keywords are used.

name — The *name* parameter specified with the CHECKNAME keyword. If pattern substitution is used, *name* is replaced with the input layer name for %_l_, and replaced with the top cell name for %_t_.

For discussion purposes, consider the following code, where layer1 and layer2 are original layers:

```
layer_global = OR layer1 layer2

RDBcheck {
    layer_local = AND layer1 layer2
    DFM RDB layer_local out.rdb <options> // n=1
    DFM RDB layer_global out.rdb <options> // n=2
}
```

The following table gives the *rulecheck_name*(s) for the preceding DFM RDB operations with different keywords specified. Assume there are two cells in the design, so *m* can take the values 1 and 2. The operations with layer_local input have n=1 and the operations with layer_global input have n=2, as in the preceding example code.

DFM RDB operation	rulecheck_name(s)
DFM RDB layer_local out.rdb	RDBcheck::<1>_1 RDBcheck::<1>_2
DFM RDB layer_global out.rdb	RDBcheck::<2>_1 RDBcheck::<2>_2
DFM RDB layer_global out.rdb CELL SPACE	RDBcheck::<2>

DFM RDB operation	rulecheck_name(s)
DFM RDB layer_local out.rdb ALL CELLS	RDBcheck::<1>
DFM RDB layer_local out.rdb CHECKNAME rdbrule	rdbrule_1 rdbrule_2
DFM RDB layer_local out.rdb CHECKNAME "%_l_"	RDBcheck::layer_local_1 RDBcheck::layer_local_2
DFM RDB layer_local out.rdb CHECKNAME "%_t_"	topcell_1 topcell_2

- *rulecheck_info*

```
flat_count hier_count num_text month day time year
```

One line containing the flat result count, the hierarchical result count, the number of check text lines that follow, and a date stamp (month day time year).

For example: 9 9 1 Nov 17 15:43:02 2016

- *check_text*

One or more lines of text.

```
[comment_string] ...
[CELL cellname cell_transform]
IL: [scope:::]layer
```

comment_string — One line for each COMMENT parameter in the DFM RDB operation. Rule check comments starting with the @ symbol are not included.

CELL cellname cell_transform — The name of the cell and six numbers defining a transform matrix that maps global coordinates to the coordinates of the cell. This line is not present if the ALL CELLS or CELL SPACE keywords are present. See DRC Cell Name for a description of the transform matrix.

IL: [scope:::]layer — The name of the input layer (IL), where *scope* is the name of the rule check that contains the DFM RDB operation and *layer* is the name of the input layer in the DFM RDB operation. The *scope::* entry is only included if the input layer is derived within the rule check (a local layer).

- *result_data*

There is a result_data block for each geometry on the input layer, with the following format:

```
result_type
result_properties
result_coordinates
```

result_type — One line with the format:

```
type ordinal num_objects
```

type — The result type, indicated by the character “p” for a polygon or “e” for an edge or error cluster.

ordinal — The ordinal of the geometry within the rule check.

num_objects — The number of objects that make up the result. For polygons, this is number of vertices, for edge and error clusters, it is the number of edges.

For example, “p 2 4” indicates a polygon which is the second object within the rule check and has four vertices.

result_properties — Result properties are included if the NODAL, ALL CELLS, or CELL SPACE keywords are used in the DFM RDB operation, or if the input layer includes DFM properties.

DN *net_number* [*net_name*]

The DN property is included if the NODAL keyword is used. The net number is given, along with the net name if it is available.

CN *cellname* *cell_transform* *flat_placement_count*

The first geometry in each cell is annotated with the CN property if the ALL CELLS or CELL SPACE keyword is included. The *cell_transform* is six numbers giving the transform matrix that maps the result coordinates to top cell space; see DRC Cell Name for a description of the transform matrix.

DFM Properties

If the input layer contains [DFM Properties](#), these are included in the result output. Cell properties (produced by the BY CELL keyword in DFM Property) have “:C” appended to the property name. Net properties (produced by the BY NET keyword in DFM Property) have “:N” appended to the property name.

result_coordinates — The coordinates for the geometry. Coordinates are supplied in top cell space using an arbitrarily selected placement of the cell.

polygon data — One line for each vertex. Each line contains one pair of coordinates.

edge data — One line for each edge. Each line contains two pairs of coordinates giving the endpoints of the edge.

Multilayer RDB File Format

Output from: A multilayer DFM RDB operation.

Contains an ASCII representation of multiple original or derived layers. Shapes on the input layers are organized into clusters. A cluster is composed of one shape from the first input layer (the primary layer) in the DFM RDB operation plus all shapes from the additional input layers (the secondary layers) that overlap the primary layer shape.

See the [DFM RDB](#) operation for information on keywords that affect the clustering behavior.

Note

 RDBs can also be generated by layer operations that include the RDB keyword, such as Density and Net Area Ratio. In such cases the RDB format is described with the command reference.

Format

The results database (RDB) is an ASCII file with the following contents:

```
header
rule_check_results
rule_check_results
...
...
```

The rule_check_results block has the following format, where the contents depend on the keywords used in the DFM RDB operation.

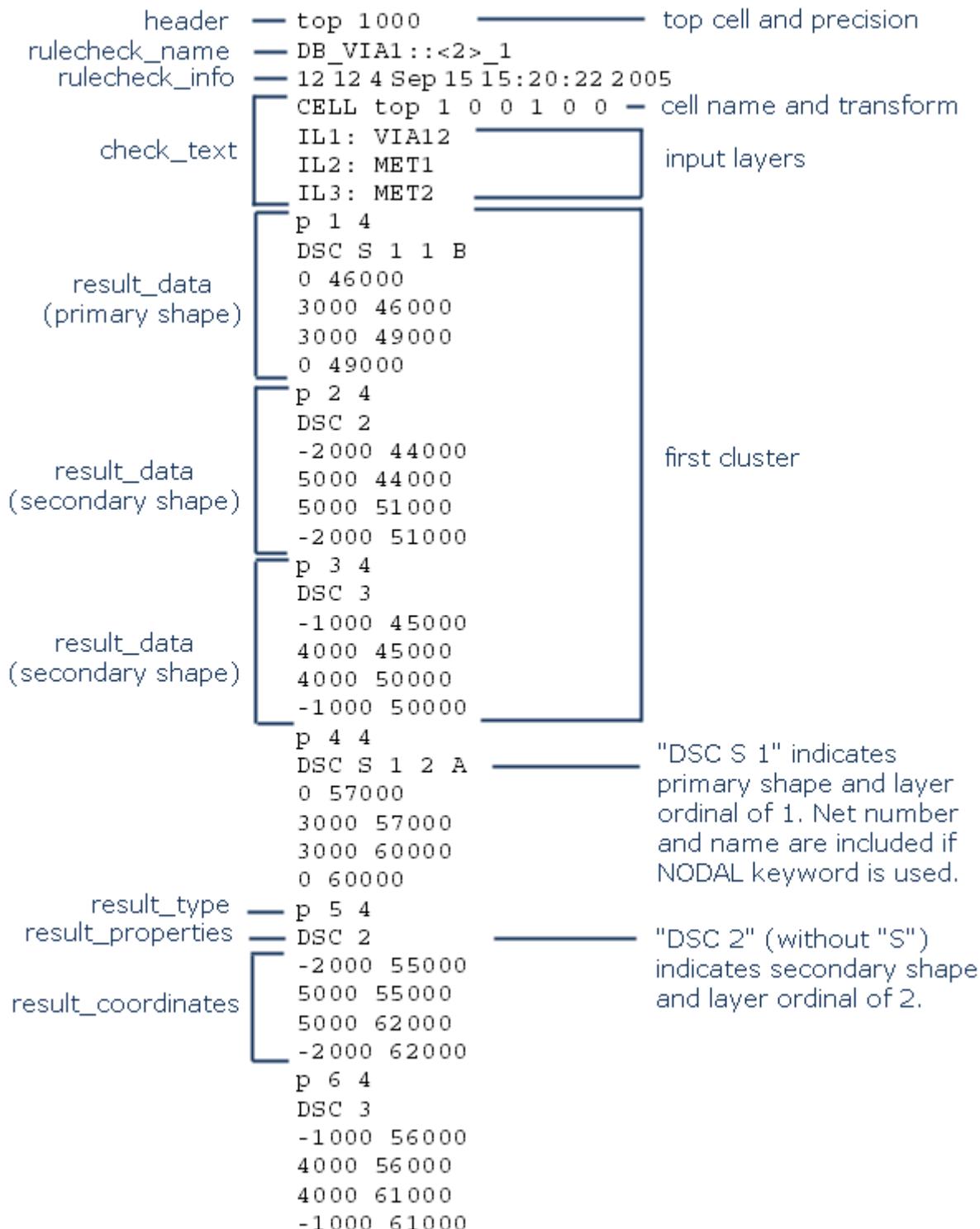
```
rulecheck_name
rulecheck_info
check_text
result_data
result_data
...
...
```

The result_data block has the following format:

```
result_type
result_properties
result_coordinates
```

The following image displays a multilayer RDB. The DFM RDB operation that created the RDB included the NODAL keyword, but not the ALL CELLS and CHECKNAME keywords. The different elements in the file format are described in the “Parameters” section following the figure.

Figure 11-32. Multilayer RDB File Format



Parameters

- *header*

The name of the top cell and the RDB precision. Example: top 1000

- *rule_check_results*

The *rule_check_results* block includes all results for a particular rule check. The block contains the rule check name, check information, rule check text, and result data. The number of *rule_check_results* blocks is determined by the ALL CELLS keyword in the DFM RDB operation:

Default — One *rule_check_results* block is written for each cell in the layout. The *check_text* block includes the CELL check text line.

ALL CELLS — A single *rule_check_results* block is written. The first geometry in each cell is annotated with the CN result property, and the *check_text* block does not include the CELL line.

- *rulecheck_name*

A string representing the rule check name. The *rulecheck_name* depends on the keywords used in the DFM RDB operation and whether the input layer was derived within the scope of the rule check containing the DFM RDB operation. The general form is the following, depending on whether CHECKNAME is used:

Table 11-5. rulecheck_name Format in Multilayer RDB

Without CHECKNAME	With CHECKNAME
<i>scope::<n>[_m]</i>	[<i>scope::</i>] <i>name[_m]</i>

scope — The name of the rule check that contains the DFM RDB operation. When CHECKNAME is specified, *scope* is only included if “CHECKNAME “%_l”” is specified and the input layer is derived within the rule check (a local layer).

n — The ordinal within the rule check of the DFM RDB operation that writes out the layer data to the RDB. The ordinal is enclosed in angle brackets (<*n*>).

m — A positive integer that uniquely identifies the cell. This integer is not used if the ALL CELLS keyword is used.

name — The *name* parameter specified with the CHECKNAME keyword. If pattern substitution is used, *name* is replaced with the first (primary) input layer name for %_l_, and replaced with the top cell name for %_t_.

For discussion purposes, consider the following code, where layer1, layer2, and layer3 are original layers:

```
layer_global = OR layer1 layer2

RDBcheck {
    layer_local = AND layer1 layer2
    DFM RDB layer_local out.rdb layer3 <options>      // n=1
    DFM RDB layer_global out.rdb layer3 <options>      // n=2
}
```

The following table gives the *rulecheck_name(s)* for the preceding DFM RDB operations with different keywords specified. Assume there are two cells in the design, so *m* can take the values 1 and 2. The operations with layer_local input have n=1 and the operations with layer_global input have n=2, as in the preceding example code.

DFM RDB operation	<i>rulecheck_name(s)</i>
DFM RDB layer_local out.rdb layer3	RDBcheck::<1>_1 RDBcheck::<1>_2
DFM RDB layer_global out.rdb layer3	RDBcheck::<2>_1 RDBcheck::<2>_2
DFM RDB layer_local out.rdb layer3 ALL CELLS	RDBcheck::<1>
DFM RDB layer_local out.rdb layer3 CHECKNAME rdbrule	rdbrule_1 rdbrule_2
DFM RDB layer_local out.rdb layer3 CHECKNAME "%_1"	RDBcheck::layer_local_1 RDBcheck::layer_local_2
DFM RDB layer_local out.rdb layer3 CHECKNAME "%_t"	topcell_1 topcell_2

- *rulecheck_info*

```
flat_count hier_count num_text month day time year
```

One line containing the flat result count, the hierarchical result count, the number of check text lines that follow, and a date stamp (month day time year).

For example: 9 9 1 Nov 17 15:43:02 2016

- *check_text*

One or more lines of text.

```
[comment_string] ...
[CELL cellname cell_transform]
ILn: [scope:::]layer
[ILn: [scope:::]layer] ...
```

comment_string — One line for each COMMENT parameter in the DFM RDB operation. Rule check comments starting with the @ symbol are not included.

CELL cellname cell_transform — The name of the cell and six numbers defining a transform matrix that maps global coordinates to the coordinates of the cell. This line is not present if the ALL CELLS keyword is present. See DRC Cell Name for a description of the transform matrix.

ILn: [scope:::]layer — The name of the input layer (IL), with these elements:

n — The ordinal of the input layer in the DFM RDB operation.

scope — The name of the rule check that contains the DFM RDB operation. The *scope::* entry is only included if the input layer is derived within the rule check (a local layer).

layer — The name of the input layer in the DFM RDB operation.

- *result_data*

There is a *result_data* block for each geometry in the rule check. The data is organized into clusters. A cluster is composed of one shape from the first input layer (the primary layer) in the DFM RDB operation plus all shapes from the additional input layers (the secondary layers) that overlap the primary layer shape. The *result_properties* line distinguishes primary and secondary shapes.

The *result_data* block has the following format:

```
result_type
result_properties
result_coordinates
```

result_type — One line with the format:

```
type ordinal num_objects
```

type — The result type, indicated by the character “p” for a polygon.

ordinal — The ordinal of the geometry within the rule check.

num_objects — The number of vertices in the polygon.

For example, “p 2 4” indicates a polygon which is the second object within the rule check and has four vertices.

result_properties — The result property DSC is always included. Other properties are included if the NODAL or ALL CELLS keywords are used in the DFM RDB operation, or if the input layer includes DFM properties.

DSC [S] *layer_ordinal* [*net_number* [*net_name*]]

The polygon descriptor, with the following entries:

S — A literal “S” is included for polygons on the primary layer. There is one such polygon per cluster and it is always the first result in each cluster.

layer_ordinal — The ordinal of the layer as it appears in the DFM RDB operation. The first polygon in each cluster, which includes the literal “S”, always has a *layer_ordinal* of 1, indicating the primary layer.

net_number [*net_name*] — Connectivity information is included if the NODAL keyword is used. The net number is given, along with the net name if it is available. The keywords NODAL, NODAL ALL, and NODAL OTHER determine if the connectivity information is included for only the first polygon in each cluster (from the primary layer), for all polygons, or for all polygons except those on the primary layer.

CN *cellname* *cell_transform* *flat_placement_count*

The first geometry in each cell is annotated with the CN property if the ALL CELLS keyword is included. The *cell_transform* is six numbers giving the transform matrix that maps the result coordinates to top cell space; see DRC Cell Name for a description of the transform matrix.

DFM Properties

If the input layer contains **DFM Properties**, these are included in the result output. Cell properties (produced by the BY CELL keyword in DFM Property) have “:C” appended to the property name. Net properties (produced by the BY NET keyword in DFM Property) have “:N” appended to the property name.

result_coordinates — The coordinates for the geometry. Coordinates are supplied in top cell space using an arbitrarily selected placement of the cell. There is one line for each vertex in the polygon, where each line contains one pair of coordinates.

Fill RDB File Format

Output from: A [DFM Fill](#) operation

Contains statistics generated by a DFM Fill operation. The Fill RDB is specified in the DFM Spec Fill statement with the RDB keyword.

The RDB groups the statistics data in rule check sections as follows:

- One rule check section if the density window, gradient window, density window step and gradient window step, are all the same size.
- Three rule check sections in all other cases.

Each rule check contains one result data block for each rectangular analysis window, as defined by the DFM Spec Fill WINDOW keyword. Statistics are reported in the results properties associated with each rule check result.

Format

The results database (RDB) is an ASCII file with the following contents:

```
header
rule_check_results
[rule_check_results] ...
```

The rule_check_results block has the following format.

```
rulecheck_name
rulecheck_info
check_text
result_data
result_data
...
```

The result_data block has the following format:

```
result_type
result_properties
result_coordinates
```

The following image displays a single check and three check fill RDB side by side, with annotations pointing out the different elements in the file structure. The elements in the file structure are described in the “Parameters” section following the figure.

Figure 11-33. Fill RDB File Format

Single Check Fill RDB

```

ccm 1000
DFM FILL SPEC ONE
40 40 1 May 16 16:00:36 2006
ILO: STAT
p 1 4
SDV 0.145696
SGV 0
WI 0
FSC1 49
FST1 49
GSW1 -1
FSC2 43
FST2 43
GSW2 -1
FSC3 47
FST3 47
GSW3 -1
FSC4 71
FST4 71
GSW4 -1
FSC5 353
FST5 353
GSW5 -1
FSC6 262
FST6 262
GSW6 -1
FSC7 3670
FST7 6267
GSW7 -1
FSC8 0
FST8 0
GSW8 -1
FDV 0.250733
FGV 0.000209857
165 210
35864 210
35864 40209
165 40209
p 2 4
SDV 0.175704
SGV 0
WI 8
FSC1 9
FST1 9
GSW1 -1
FSC2 16
:
:
remaining result_properties
and result_coordinates
are not shown

```

Three Check Fill RDB

```

header
rulecheck_name
rulecheck_info
check_text
result_data
block
result_type
result_properties
(statistics)
result_coordinates
(vertices of the
analysis area)
rulecheck_name
(second rule check)
rulecheck_name
(third rule check)
mm6 1000
DFM FILL SPEC three
144 144 1 Mar 1 12:53:57 2007
ILO: STAT
p 1 4
WI 1
FSC1 3
FST1 4
FSTAT OK
0 0
12499 0
12499 12499
0 12499
p 2 4
WI 2
FSC1 4
FST1 6
FSTAT OK
0 12500
12499 12500
12499 24999
0 24999
:
:
rulecheck_name DFM FILL SPEC three DENSITY
144 144 1 Mar 1 12:53:57 2007
ILO: DENSITY
p 1 4
SDV 0
FDV 0.608
FSTAT OK
0 0
49999 0
49999 49999
0 49999
:
:
rulecheck_name DFM FILL SPEC three GRADIENT
36 36 1 Mar 1 12:53:57 2007
ILO: GRADIENT
p 1 4
WI 1
SGV -1
SDV 0
GSW1 7
FGV 0.084
FDV 0.608
FSTAT OK
0 0
49999 0
49999 49999
0 49999

```

Parameters

- *header*

The name of the top cell and the RDB precision. Example: top 1000

- *rule_check_results*

The statistics are organized in *rule_check_results* blocks. The block contains the rule check name, check information, rule check text, statistics, and analysis window coordinates. The number of *rule_check_results* blocks is determined by the parameters in the DFM Spec Fill statement:

One rule check — One rule check section if the density window, gradient window, density window step, and gradient window step are all the same size.

Three rule checks — Three rule check sections in all other cases.

If multiple optimizers are used, this changes the number of rule check sections, as explained with the definition of the *rulecheck_name*.

- *rulecheck_name*

The *rulecheck_name* depends on whether there are one or three *rule_check_results* sections and the number of optimizers.

One rule check section, one optimizer:

DFM FILL SPEC *name* — This rule check reports all meaningful statistics.

Three rule check sections, one optimizer:

DFM FILL SPEC *name* — This rule check reports statistics for fill shape counts. It contains results for each of the subwindows of the common grid for the various window options.

DFM FILL SPEC *name* DENSITY — This rule check reports statistics on DENSITY and MAGNITUDE constraints. It contains analysis windows defined by the WINDOW... STEP options.

DFM FILL SPEC *name* GRADIENT — This rule check reports statistics on GRADIENT constraints. It contains analysis windows defined by GWINDOW...STEP options.

When multiple optimizers are specified for fill generation, this is indicated by adding the “GRID# *n*” field to the *rulecheck_name*. Different rule checks are generated according to the number of optimizers and how the window size and step is specified for each optimizer specification. The ordinal *n* is incremented for each rule check name.

The following examples illustrate the rule check naming convention with multiple optimizers which reference the same fill spec.

Two optimizers with identical STEP and WINDOW:

Optimizer 1: DFM FILL SPEC *name* (GRID# 1)

Optimizer 2: DFM FILL SPEC *name* (GRID# 2)

Two optimizers, where optimizer 1 has a different STEP size from the WINDOW size:

- Optimizer 1: DFM FILL SPEC *name* (GRID# 1)
- Optimizer 1: DFM FILL SPEC *name* DENSITY (GRID# 2)
- Optimizer 1: DFM FILL SPEC *name* GRADIENT (GRID# 3)
- Optimizer 2: DFM FILL SPEC *name* (GRID# 4)

Two optimizers, both with different STEP and WINDOW sizes

- Optimizer 1: DFM FILL SPEC *name* (GRID# 1)
- Optimizer 1: DFM FILL SPEC *name* DENSITY (GRID# 2)
- Optimizer 1: DFM FILL SPEC *name* GRADIENT (GRID# 3)
- Optimizer 2: DFM FILL SPEC *name* (GRID# 4)
- Optimizer 2: DFM FILL SPEC *name* DENSITY (GRID# 5)
- Optimizer 2: DFM FILL SPEC *name* GRADIENT (GRID# 6)

- *rulecheck_info*

```
flat_count hier_count num_text month day time year
```

One line containing the flat result count, the hierarchical result count, the number of check text lines that follow, and a date stamp (month day time year). The result count gives the number of analysis windows for the rule check.

For example: 9 9 1 Nov 17 15:43:02 2016

- *check_text*

One line of text.

```
ILO: {STAT | DENSITY | GRADIENT}
```

ILO: STAT — When the RDB contains one rule check, all statistics are reported in the corresponding rule check. When the RDB contains three rule checks, only fill statistics are reported in the rule check.

ILO: DENSITY — The rule check reports density statistics for the run. This is used in the second rule check when the RDB contains three rule checks.

ILO: GRADIENT — The rule check reports gradient statistics for the run. This is used in the third rule check when the RDB contains three rule checks.

- *result_data*

There is one *result_data* block for each analysis area. The analysis area is always a rectangle, regardless of the type of layer on which the analysis was performed, and represents either a cell extent or a capture window, as defined by the analysis controls in the DFM Spec Fill statement. The *result_data* block has the following format:

```
result_type
result_properties
result_coordinates
```

result_type — One line with the format:

```
type ordinal num_objects
```

type — The result type, indicated by the character “p” for a polygon.

ordinal — The ordinal of the analysis area within the rule check.

num_objects — The vertex count, which is always 4.

For example, “p 2 4” indicates a polygon which is the second analysis area within the rule check and has four vertices.

result_properties — There are multiple lines reporting the fill statistics. There is one line per fill statistic, reported as a property name followed by its value. The statistics are divided among the different rule checks as described in the *rulecheck_name* definition. The following statistics are reported:

Table 11-6. DFM Fill Statistics

Property Name	Description
SDV or SDV_<opt>	Starting Density Value. Density of the window as calculated by the optimizer <opt> before any fill was added to it.
SGV or SGV_<opt>	Starting Gradient Value. Absolute value of the largest negative density gradient to the neighbor windows as calculated by the optimizer <opt> before any fill was added to the design. This value can be different from what DENSITY reports because it only calculates gradient from higher-density windows to lower-density ones (hence “negative” gradient). If window does not have neighbors with lower density, the SGV value is reported as -1.
WI	Index of the window.
FSC1, FSC2, ... or FSC_<opt>_1, FSC_<opt>_2, ...	Fill Shape Count. This is the total number of fill shapes added to the window. A property is included for each optimizer and fill shape specified in the operation.
GSW1, GSW2,... or GSW_<opt>_1, GSW_<opt>_2,...	Gradient Source Window. If target density of the window was adjusted because of the gradient constraint, this property contains an index of the window that caused the gradient adjustment. A property is included for each fill shape and optimizer specified in the operation.
FDV or FDV_<opt>	Final Density Value. Density of the window as calculated by the optimizer <opt> after all fill was added to the window.

Table 11-6. DFM Fill Statistics (cont.)

Property Name	Description
FGV or FGV_<opt>	Final Gradient Value. Absolute value of the largest negative density gradient to the neighboring windows as calculated by the optimizer <opt> after all fill was added to the design. This value can be different from what DENSITY reports because it only calculates gradient from higher-density windows to lower-density ones (hence “negative” gradient). If window does not have neighbors with lower density, the FGV value is reported as -1.
FST	Fill Shape Total: The number of fill shapes that were left after the optimizer.
FSTAT or FSTAT_<opt>	Status of the window after all fill shapes were added to the design. The property can have one or more of the following values: <ul style="list-style-type: none"> • OK — Window passes all constraints. • GRAD — Window failed gradient constraint. • MIN — Window failed minimum density constraint. • MAX — Window failed maximum density constraint. • MAG — Window failed magnitude constraint. One or more failed statuses can be concatenated in a string, for example "MIN MAG GRAD" means the window failed minimum density constraint, magnitude and gradient.

result_coordinates — The coordinates for the analysis area. There is one line for each vertex, and each line contains one pair of coordinates.

DFM File Formats

DFM operations use various file formats.

Table 11-7 provides a summary of file formats used by DFM operations.

Table 11-7. File Formats Used by DFM Operations

File	Description
Histogram File	Output file from DFM Histogram . Contains histogram data.
CAA Defect Data Format	Input file for performing critical area analysis.
CAA Layers File	File format for mapping user specific layer names to Mentor Standard Layer Names .
Fill Layermap File	File format for mapping layer names in a DFM database to layer names in the original layout. For use when using backannotation from a DFM database.

Histogram File

This section describes the histogram file.

The histogram file contains histogram data generated by [DFM Histogram](#).

Organization: One or more sets of histogram data. There is one set of data for each DFM Histogram operation that writes data to this file.

Each set of histogram data is organized into a header, followed by one histogram per cell.

Header

Two or more lines.

- Name of the top cell, followed by the precision.
- The word “PROPERTY” followed by the name of the property this histogram reports on, followed by the name(s) of the layer or layers from which the properties were extracted.
- One line for each comment specified in the DFM Histogram operation that created this file.

Histogram

Two lines of histogram summary followed by one line per histogram bin.

- Summary line 1: The bin report. This line reports the following:
 - **bin_count BINS** — The actual number of bins, whether they were specified by the user or computed.
 - **MIN min_value(below-min-count)** — The minimum value reported, as defined in the DFM Histogram operation. Following that is the number of results that were below *min_value* and therefore not included in the histogram.
 - **MAX max_value(above-max-count)** — The maximum value reported, as defined in the DFM Histogram operation. Following that is the number of results that were above the *max_value* and therefore not included in the histogram.
- Summary line 2: The cell report. This line reports the following:
 - **CELL cell_name** — The name of the cell whose properties are reported in this histogram.
 - The method used to collect the data:
 - (ACCUMULATED) means the histogram reports the results of all levels of the specified cell. That is, all cells referenced within the specified cell are counted.
 - (FLAT) means the histogram reports the results for the entire chip and not in BY CELL mode.
 - Neither means the histogram reports on the top cell only.
- Bin Data: one line per bin. Each line contains four values:
 - **Bin Center** — The median value for the bin.
 - **Result Count** — The number of geometries on the input layers whose property values fall into the current bin.
 - **Normalized Count** — The total number of geometries, normalized so that the total area under the histogram is one.
 - **Result Fraction** — The fraction of the results in this bin (the result count divided by the total result count in all bins).

Example 11-1. DFM Histogram

The following code finds locations where two polygons on layers MET1 or MET2 are closely spaced and computes the distribution of lengths of these areas:

```
MET1S = EXT MET1 < 0.25
MET2S = EXT MET2 < 0.2
MET1SE = DFM PROPERTY MET1S [ EC = EC( MET1S ) ]
MET2SE = DFM PROPERTY MET2S [ EC = EC( MET2S ) ]
CHECK {
    DFM HISTOGRAM "hist1_2h" MET1SE MET2SE PROPERTY "EC" BY CELL
    ACCUMULATE BINS 5
}
```

Figure 11-34. Sample Histogram File

The diagram illustrates a sample histogram file structure. It shows three distinct histograms, each with its own header and data. The first histogram is for the 'top' cell, with a header line 'top 1000'. The second histogram is for 'cell11 (ACCUMULATED)', and the third is for 'cell12 (ACCUMULATED)'. Each histogram has a header line starting with '5 BINS: MIN 0(0) MAX 1.9(0)' followed by a list of five bins. Each bin entry consists of a center value, a count, and a norm_count fraction. A callout box labeled 'One histogram per cell.' spans all three histograms.

```
top 1000
PROPERTY EC: MET1SE MET2SE
5 BINS: MIN 0(0) MAX 1.9(0)
CELL cell11 (ACCUMULATED)
0.19: 3 0.56391 0.214286
0.57: 3 0.56391 0.214286
0.95: 0 0 0
1.33: 8 1.50376 0.571429
1.71: 0 0 0
5 BINS: MIN 0(0) MAX 1.9(0)
CELL cell12 (ACCUMULATED)
0.19: 3 0.657895 0.25
0.57: 5 1.09649 0.416667
0.95: 2 0.438596 0.166667
1.33: 2 0.438596 0.166667
1.71: 0 0 0
5 BINS: MIN 0(0) MAX 1.9(0)
CELL top(ACCUMULATED)
0.19: 20 0.93985 0.357143
0.57: 11 0.516917 0.196429
0.95: 5 0.234962 0.0892857
1.33: 18 0.845865 0.321429
1.71: 2 0.093985 0.0357143
```

CAA Defect Data Format

The CAA defect data contains a set of particle sizes and density lists for a particular process.

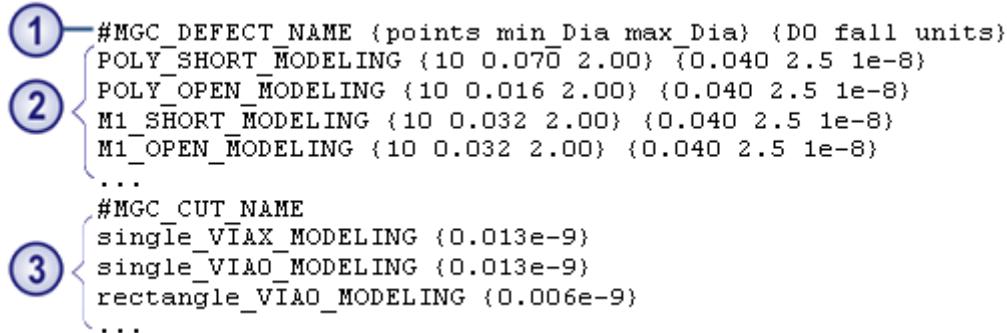
In general, this varies from layer to layer and according to the critical area type: OPEN or SHORT.

Note

 Every foundry or IDM has their own defect data format, and many of them are encrypted prior to distribution. The Calibre Critical Area Analysis tool accepts most encrypted data file formats. Critical area rule decks created using encrypted defect data are also encrypted, ensuring that defect data cannot be extracted from the generated code.

If you do not have a foundry-supplied defect data file, you can create your own using the MGC Defect Data format. In this format, layer, defect type, and range of defect sizes are followed by defect density. Commented lines are ignored. [Figure 11-35](#) shows a truncated CAA defect data file in the D0 Defect Data format.

Figure 11-35. CAA Defect Data Format Explanation



```
#MGC_DEFECT_NAME (points min_Dia max_Dia) {D0 fall units}
POLY_SHORT_MODELING (10 0.070 2.00) {0.040 2.5 1e-8}
POLY_OPEN_MODELING (10 0.016 2.00) {0.040 2.5 1e-8}
M1_SHORT_MODELING (10 0.032 2.00) {0.040 2.5 1e-8}
M1_OPEN_MODELING (10 0.032 2.00) {0.040 2.5 1e-8}
...
#MGC_CUT_NAME
single_VIAX_MODELING {0.013e-9}
single_VIAO_MODELING {0.013e-9}
rectangle_VIAO_MODELING {0.006e-9}
...
```

Where the content of the numbered sections is as follows:

1. Commented line that shows the Power Modeling Format of the Defect Type.
2. Layer and Defect Type, Defect Range, and Defect Density.
3. Via and Contact Failure Rates.

When creating a CAA defect data file, you use one of two formats, the Power Modeling Format or the User-Defined List Format. The difference between the formats is the layers' defect data format.

- **Power Modeling Format** — The following examples show CAA defect files that contain the MODELING keyword. When using the Power Modeling Format, note that there are three different MGC Defect Data Formats dependent on the version of the Calibre CAA tool and the type of defect density data.

- D0 Defect Data Format — [Example 11-2](#) shows a CAA D0 defect data file using this format (supported for Calibre CAA tool versions 2015.4 and later). You can use this format for DFM yield analysis when you have foundry defect density (D0) data available.
- No Defect Density (NDD) Data Format — [Example 11-3](#) shows a CAA NDD defect data file using this format (supported for Calibre CAA tool versions 2017.2 and later). You can use this format for DFM scoring when you do not have foundry defect density (D0) data available.
- Mentor Graphics K-Factor Defect Data Format — [Example 11-4](#) shows a CAA K-Factor defect data file using this format (supported for all Calibre CAA tool versions). You can use this format for DFM yield analysis.

Note

 You cannot mix the Power Modeling Defect Data Formats in the same CAA defect data file.

- **User-Defined List Format** — [Example 11-5](#) shows a CAA defect file in the User-Defined List Format. The layer names contain the USER_DEFINED keyword. When using this format, note that on each line the range of defect sizes and the list of densities must have the same number of list entries. The order of the lines is not important.

Note

 You cannot mix the Power Modeling Format and the User-Defined List Format in the same CAA defect data file.

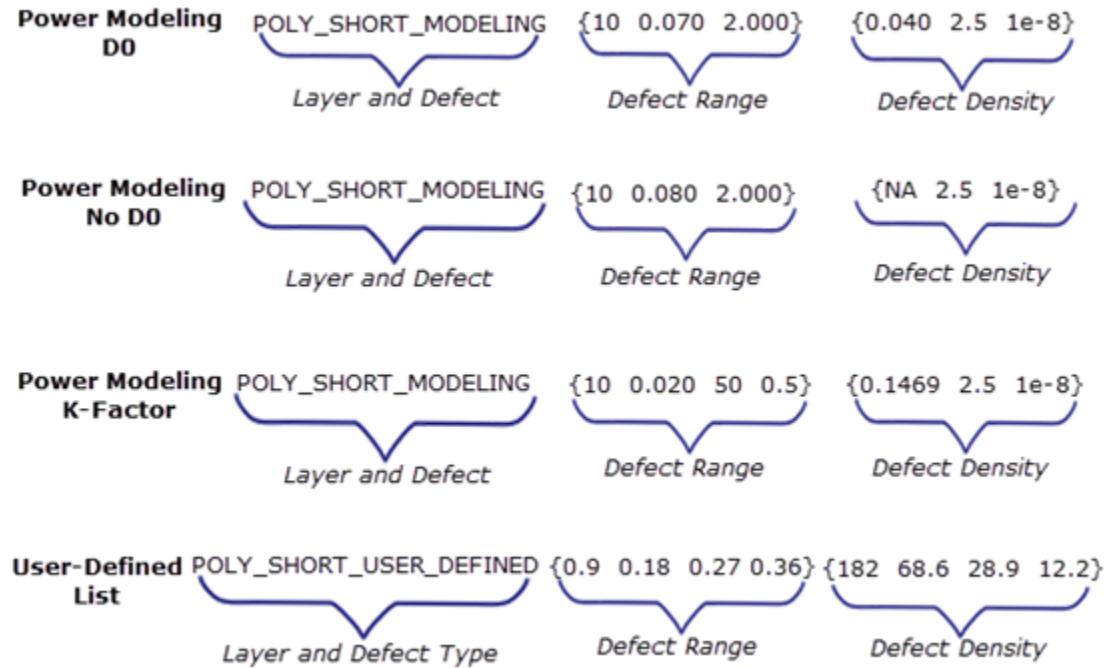
Format (Defect Data)	351
Layer and Defect Type, Defect Range and Defect Density Data	352
Layer Names	363

Format (Defect Data)

The format of the defect data file is one entry per line that provides the defect layer name and type, range of defect sizes, and defect density specification.

Defect Data Formats

Figure 11-36. Power Modeling and User-Defined List Defect Data Formats



For all entries, comment text is denoted using the pound sign (#). Unless specified otherwise, all elements for a given data format are mandatory.

Layer and Defect Type, Defect Range and Defect Density Data

Modeling defect types requires certain layer and defect parameter specifications. The CAA defect file contains one set of data per line that is used to specify the defect model type, the layer information, and the defect parameters.

Each represents the data per layer and critical area type as in the following example:

- **Layer and Defect Type (Shorts and Opens)** — Use one of the following:
 - **Power Modeling Format** — Each line starts with the list name in the following format:
layer_type_MODELING
 - *layer* — The layer name. Some layer names may be user-defined, while others are predefined. Refer to “[Layer Names](#)” on page 363 for details.
 - *type* — Defect type. Either SHORT or OPEN.

- **MODELING** — A mandatory literal string.
- **User-Defined List Format** — Each line starts with the list name in the following format:

layer_type_USER_DEFINED

 - *layer* — The layer name. Some layer names may be user-defined, while others are predefined. Refer to “[Layer Names](#)” on page 363 for details.
 - *type* — Either SHORT or OPEN.
 - **USER_DEFINED** — A mandatory literal string.
- **Defect Range** — A range of defect sizes for analysis. Use one of the following formats:
 - **Power Modeling (D0 Defect Data Format)** — A 3-tuple set of values containing the parameters required to generate the range of defect sizes using the D0 defect size model (supported for Calibre CAA tool versions 2015.4 and later) specified as follows:
 - Number of defects.
 - Lower limit on defect size (corresponds to a minimum defect diameter specified in user units, typically microns).
 - Upper limit on defect size (corresponds to a maximum defect diameter specified in user units, typically microns).
 - **Power Modeling (No Defect Density Format)** — A 3-tuple set of values containing the parameters required to generate the range of defect sizes (supported for Calibre CAA tool versions 2017.2 and later) specified as follows:
 - Number of defects.
 - Lower limit on defect size (corresponds to the design rule’s minimum critical dimension (CD) specified in user units, typically microns).
 - Upper limit on defect size (corresponds to a user-defined (usually arbitrary) maximum defect diameter specified in user units, typically microns).
 - **Power Modeling (K-Factor Defect Data Format)** — A 4-tuple set of values containing the parameters required to generate the range of defect sizes using the Mentor Graphics embedded defect size model (supported for all Calibre CAA tool versions) specified as follows:
 - Number of defects.
 - Minimum CD (corresponds to a defect diameter).
 - Upper limit on defect size (a multiplier applied to the minimum CD to obtain a maximum defect radius).

- Lower limit on defect size (a multiplier applied to the minimum CD to obtain a minimum defect radius). This value is normally 0.5
- **User-Defined List Format** — A list of sizes. See “[Defect Data File \(User-Defined List Format\)](#)” on page 363.
- **Defect Density** — Use one of the following defect density formats:
- **Power Modeling (D0 Defect Data Format)** — A 3-tuple set of values containing parameters required to generate the defect density list using the power model and the D0 normalized defect density formula (supported for Calibre CAA tool versions 2015.4 and later). Units are defects/cm².

The normalized defect density means that the integral of the defect density over the diameter range = 1.

$$\int_{dmin}^{dmax} f(x)dx = 1$$

For a diameter range (a,b) , the defect density is defined by the fall power (n) as shown.

$$D_{(a,b)} = D0 \int_a^b f(x)dx$$

$$D0 = D_{(a,b)} \frac{1}{(n-1)} \left(\frac{1}{dmin^{n-1}} - \frac{1}{dmax^{n-1}} \right)$$

Where:

- **D0** — Defect density constant for a normalized defect density distribution.
- **$D(a,b)$** — Defect density distribution over a particle diameter range defined by (a, b) .
- **$dmin$** — Minimum diameter size for a defect to be considered for analysis.
- **$dmax$** — Maximum diameter size for a defect to be considered for analysis.
- **n** — Fall power exponent in the power model of defect density versus defect size.
- **Conversion Factor** — For instance, to convert from um² to cm².
- **Power Modeling (No Defect Density Format)** — A 3-tuple set of values containing parameters required to generate the defect density list using the power

model and the Avg_Quality metric for DFM scoring of layers and rules. Results are normalized for design comparison checking (supported for Calibre CAA tool versions 2017.2 and later). Units are defects/cm. See “[CAA NDD Metrics](#)” on page 118.

- “**NA**” — Replaces the specification for the D0 defect density value(s).
- **Fall Power** — Can be specified as a global number.
- **Conversion Factor** — For instance, to convert from um^2 to cm^2 .
- **Power Modeling (K-Factor Defect Data Format)** — A 3-tuple set of values containing parameters required to generate the defect density list using the power model and the K-Factor defect density formula (supported for all Calibre CAA tool versions).

$$\text{defect_density} = k/(x^n)$$

Where:

- **k** — Defect density constant for power model defect distribution $D(x)=k/(x^n)$ where $D(x)$ is defect density in $1/\text{cm}^2$, x is defect size (radius) in um .
- **n** — Fall power exponent in the power model of defect density versus defect size.
- **Conversion Factor** — For instance, to convert from um^2 to cm^2 .
- **User-Defined List Format** — A list of densities. See “[Defect Data File \(User-Defined List Format\)](#)” on page 363.
- **Inter-Layer Short (ILS) Modeling** — To define inter-layer short defects, use the following format:

ILS_MODELING *layer1 layer2 {density}*

 - **ILS_MODELING** — A mandatory literal string.
 - **layer1 layer2** — The layer names. Short defects are detected between these specified layers.
 - **density** — Enclose in brackets {} and calculate using the following formula:

$\text{density} = \text{defects}/\text{um}^2$
- **Failure Rates** — One row for each contact or via layer, defining the failure rates for single contacts on the contact layer, single or isolated vias on a via layer, or rectangular vias on a via layer.

These lines are specified with **CONT** or **VIAx** for type instead of SHORT or OPEN, and are prefixed with **single_**, **iso_**, **large_**, or **rectangle_**. For example:

```
single_OD_CONT_MODELING
single_POLY_CONT_MODELING
single_NDIFF_CONT_MODELING
single_PDIFF_CONT_MODELING
single_BUTTED_CONT_MODELING
rectangle_VIAx_MODELING
iso_VIAx_MODELING
large_VIAx_MODELING
rectangle_VIAx_MODELING
rectangle_VIA0_MODELING
single_VIAx_MODELING
```

The last row (single_VIAx_MODELING) of the example contains a failure rate list for via1...vian. If this row contains only one value, that value is used for all via layers. The order of the lines is not important.

If you choose to supply multiple via failure rates for single_VIAx_MODELING and the number of entries is less than the number of specified vias, Calibre Interactive displays a message prompt informing you of this condition. If you accept the popup dialog, the last entry is used for the remaining vias, and the analysis continues. In batch mode, a message is displayed, but no prompt. This behavior is true for all via type analysis which support multi fail rate format.

If you are using the CAA NDD flow for DFM scoring (supported for Calibre CAA tool versions 2017.2 and later), all via failure rate values are replaced with “NA”, and an Avg_Quality metric is used based on the single via percentage (*Single_Via%*). See “[CAA NDD Metrics](#)” on page 118.

- **Open Via Defects** — You can specify VIA0 and VIAx Open Via defects for CAA with the same [Defect Data Formats](#) used for OPEN and SHORT defect types. For example,

- Power Modeling (D0 Defect Data Format)

```
VIA0_MODELING {Defect Range} {Defect Density}
```

- Power Modeling (No Defect Density Format with *D0* values specified as “NA”)

```
VIA0_MODELING {Defect Range} {Defect Density}
```

- Power Modeling (K-Factor Defect Data Format)

```
VIA0_MODELING {Defect Range} {Defect Density}
```

- User-Defined List Format

```
VIA0_USER_DEFINED {Defect Range} {Defect Density}
```

- **Isolated Via Defects** — For isolated via defects (iso_VIAx_MODELING) the defect data is defined with the following parameters:
 - distance*** — The isolation rule distance in um. This is the first number specified.
 - fail_rate*** — The fail rate(s). These are the remaining numbers specified after the isolation distance. Any via without neighbors falling within the specified distance is subject to the failure rate. Otherwise, the single via failure rate applies.

There are two formats for specifying isolated via defects:

- iso_VIAx_MODELING {***distance fail_rate ...***}

In this format, the same distance value applies to all via layers. If only one ***fail_rate*** number is specified, it applies to all via layers. If multiple ***fail_rate*** numbers are specified, there must be enough fail rates specified to match all via layers in the layer map.

The following example shows how to specify the isolated via defect using the same distance format:

```
iso_VIAx_MODELING {2 5e-2 5e-3 5e-4 5e-5 4e-4}
```

- iso_VIAx_MODELING {{***distance fail_rate ...***} {***distance fail_rate ...***} ...}

In this format, multiple distance values are specified with the associated set of fail rates. There must be enough fail rates specified to match all via layers in the layer map. This format allows for specification of large vias on upper layers to use different isolation distances than small vias on lower layers.

The following example shows how to specify the isolated via defect using the multiple distance format:

```
iso_VIAx_MODELING {{2 5e-2 5e-3} {3 5e-4 5e-5 4e-4}}
```

Where:

{}{ 2 ...} — The distance for VIA1 and VIA2.

{ 3 ...} — The distance for VIA3, VIA4, VIA5, and so on.

- **Large Via Defects** — For large via defects (large_VIAx_MODELING), the following apply:
 - Use the iso_VIAx_MODELING format for this type of defect definition. For example:

```
large_VIAx_MODELING {2 1e-9 2e-9 3e-9 5e-9}
```

indicates that the via size threshold is 2 um X 2 um, followed by the fail rates for defined vias.

- The via size is checked in both the x and y direction. If the aspect ratio of the via is not 1, then it is considered a rectangular via and that defect type applies.
- If large_VIAx_MODELING is specified, then large via defect checking is excluded from rectangular via and single via checks.
- For multiple fail rates, use the following format:

```
large_VIAx_MODELING {{2 5e-2 5e-3} {3 5e-4 5e-5 4e-4}}
```

Where:

`{{2 ...}}` — The size for VIA1 and VIA2.

`{3 ...}` — The size for VIA3, VIA4, VIA5, and so on.

Vias and contacts density MODELING and USER_DEFINED formats are the same.

Table 11-8 summarizes the supported defect types.

Table 11-8. Supported Defect Types Summary

Defect Type
CA_LI_OPEN_MODELING
CA_LI_SHORT_MODELING
CAB_LI_OPEN_MODELING
CAB_LI_SHORT_MODELING
CAPC_LI_SHORT_MODELING
CB_LI_OPEN_MODELING
CB_LI_SHORT_MODELING
DIFF_OPEN_MODELING
DIFF_SHORT_MODELING
ILS_MODELING
iso_CONT_MODELING
iso_VIAx_MODELING
large_VIAx_MODELING
<i>metal_layer_name_OPEN_MODELING</i>
<i>metal_layer_name_SHORT_MODELING</i>
NDIFF_CONT_MODELING
OD_CONT_MODELING
ODFIN_OPEN_MODELING

Table 11-8. Supported Defect Types Summary (cont.)

Defect Type
ODFIN_SHORT_MODELING
PDIFF_CONT_MODELING
POLY_CONT_MODELING
POLY_OPEN_MODELING
POLY_SHORT_MODELING
rectangle_VIA0_MODELING
rectangle_VIAX_MODELING
SD_SHORT_MODELING
single_BUTTED_CONT_MODELING
single_CA_LI_MODELING
single_CB_LI_MODELING
single_NDIFF_LI_MODELING
single_OD_CONT_MODELING
single_PDIFF_LI_MODELING
single_POLY_CONT_MODELING
single_TS_LI_MODELING
single_VIA0_MODELING
single_VIAX_MODELING
TS_LI_OPEN_MODELING
TS_LI_SHORT_MODELING
TSCA_LI_SHORT_MODELING
TSCB_LI_SHORT_MODELING
TSPC_LI_SHORT_MODELING
VIA0_MODELING
VIAX_MODELING

[Example 11-2](#) on page 360 shows the contents of an example CAA defect file using the D0 Power Modeling Format.

Example 11-2. D0 Defect File (Power Modeling Format)

```
#MGC_DEFECT_NAME {pts min_Dia max_Dia} {D0 fall units}
ODFIN_SHORT_MODELING {10 0.040 0.40} {0.040 2.5 1e-8}
POLY_SHORT_MODELING {10 0.070 2.00} {0.040 2.5 1e-8}
CAPC_LI_SHORT_MODELING {10 0.016 0.16} {0.040 2.5 1e-8}
CA_LI_SHORT_MODELING {10 0.050 2.00} {0.040 2.5 1e-8}
CB_LI_SHORT_MODELING {10 0.058 2.00} {0.040 2.5 1e-8}
M1_SHORT_MODELING {10 0.032 2.00} {0.040 2.5 1e-8}
M2_SHORT_MODELING {10 0.032 2.00} {0.040 2.5 1e-8}
M3_SHORT_MODELING {10 0.032 2.00} {0.040 2.5 1e-8}
M4_SHORT_MODELING {10 0.040 2.00} {0.040 2.5 1e-8}
M5_SHORT_MODELING {10 0.040 2.00} {0.040 2.5 1e-8}
M6_SHORT_MODELING {10 0.040 2.00} {0.040 2.5 1e-8}
M7_SHORT_MODELING {10 0.040 2.00} {0.040 2.5 1e-8}
M8_SHORT_MODELING {10 0.062 2.00} {0.040 2.5 1e-8}
M9_SHORT_MODELING {10 0.062 2.00} {0.040 2.5 1e-8}
M10_SHORT_MODELING {10 0.126 2.00} {0.040 2.5 1e-8}
M11_SHORT_MODELING {10 0.126 2.00} {0.040 2.5 1e-8}
##
ODFIN_OPEN_MODELING {10 0.020 0.20} {0.040 2.5 1e-8}
POLY_OPEN_MODELING {10 0.016 2.00} {0.040 2.5 1e-8}
CA_LI_OPEN_MODELING {10 0.040 2.00} {0.040 2.5 1e-8}
CB_LI_OPEN_MODELING {10 0.040 2.00} {0.040 2.5 1e-8}
M1_OPEN_MODELING {10 0.032 2.00} {0.040 2.5 1e-8}
M2_OPEN_MODELING {10 0.032 2.00} {0.040 2.5 1e-8}
M3_OPEN_MODELING {10 0.032 2.00} {0.040 2.5 1e-8}
M4_OPEN_MODELING {10 0.040 2.00} {0.040 2.5 1e-8}
M5_OPEN_MODELING {10 0.040 2.00} {0.040 2.5 1e-8}
M6_OPEN_MODELING {10 0.040 2.00} {0.040 2.5 1e-8}
M7_OPEN_MODELING {10 0.040 2.00} {0.040 2.5 1e-8}
M8_OPEN_MODELING {10 0.062 2.00} {0.040 2.5 1e-8}
M9_OPEN_MODELING {10 0.062 2.00} {0.040 2.5 1e-8}
M10_OPEN_MODELING {10 0.126 2.00} {0.040 2.5 1e-8}
M11_OPEN_MODELING {10 0.126 2.00} {0.040 2.5 1e-8}
##
# MGC_CUT_NAME single_cut_failure_rate(s)
single_CB_LI_MODELING {0.013e-9}
single_CA_LI_MODELING {0.010e-9}
single_VIAx_MODELING {0.013e-9}
single_VIA0_MODELING {0.013e-9}
rectangle_VIA0_MODELING {0.006e-9}
rectangle_VIAx_MODELING {0.006e-9}
```

[Example 11-3](#) on page 361 shows the contents of an example CAA defect file using the No Defect Density (NDD) Power Modeling Format. The D0 defect density value(s) and via single_cut_failure_rate(s) are replaced with “NA”.

Example 11-3. No Defect Density (NDD) (Power Modeling Format)

```
#MGC_DEFECT_NAME {points min max} {D0 fall units}
DIFF_SHORT_MODELING {10 0.080 2.000} {NA 2.5 1e-8}
POLY_SHORT_MODELING {10 0.080 2.000} {NA 2.5 1e-8}
METAL1_SHORT_MODELING {10 0.070 2.000} {NA 2.5 1e-8}
METAL2_SHORT_MODELING {10 0.080 2.000} {NA 2.5 1e-8}
METAL3_SHORT_MODELING {10 0.080 2.000} {NA 2.5 1e-8}
METAL4_SHORT_MODELING {10 0.080 2.000} {NA 2.5 1e-8}
METAL5_SHORT_MODELING {10 0.080 2.000} {NA 2.5 1e-8}
METAL6_SHORT_MODELING {10 0.080 2.000} {NA 2.5 1e-8}
METAL7_SHORT_MODELING {10 0.080 2.000} {NA 2.5 1e-8}
METAL8_SHORT_MODELING {10 0.080 2.000} {NA 2.5 1e-8}
METAL9_SHORT_MODELING {10 0.140 2.000} {NA 2.5 1e-8}
METAL10_SHORT_MODELING {10 0.140 2.000} {NA 2.5 1e-8}
DIFF_OPEN_MODELING {10 0.060 2.000} {NA 2.5 1e-8}
POLY_OPEN_MODELING {10 0.040 2.000} {NA 2.5 1e-8}
METAL1_OPEN_MODELING {10 0.070 2.000} {NA 2.5 1e-8}
METAL2_OPEN_MODELING {10 0.080 2.000} {NA 2.5 1e-8}
METAL3_OPEN_MODELING {10 0.080 2.000} {NA 2.5 1e-8}
METAL4_OPEN_MODELING {10 0.080 2.000} {NA 2.5 1e-8}
METAL5_OPEN_MODELING {10 0.080 2.000} {NA 2.5 1e-8}
METAL6_OPEN_MODELING {10 0.080 2.000} {NA 2.5 1e-8}
METAL7_OPEN_MODELING {10 0.080 2.000} {NA 2.5 1e-8}
METAL8_OPEN_MODELING {10 0.080 2.000} {NA 2.5 1e-8}
METAL9_OPEN_MODELING {10 0.140 2.000} {NA 2.5 1e-8}
METAL10_OPEN_MODELING {10 0.140 2.000} {NA 2.5 1e-8}

# MGC_CUT_NAME single_cut_failure_rate(s)
single_POLY_CONT_MODELING {NA}
single_BUTTED_CONT_MODELING {NA}
single_OD_CONT_MODELING {NA}
single_VIAx_MODELING {NA}
```

[Example 11-4](#) on page 362 shows the contents of an example CAA defect file using Mentor Graphics K-Factor Power Modeling Format.

Example 11-4. K-Factor Defect File (Power Modeling Format)

```
#MGC_DEFECT_NAME {points min_CD max_mult min_mult} {Kfactor fall units}
DIFF_SHORT_MODELING {10 0.028 35.71 0.5} {0.1469 2.0 1e-8}
DIFF_OPEN_MODELING {10 0.028 35.71 0.5} {0.1469 2.0 1e-8}
POLY_SHORT_MODELING {10 0.020 50 0.5} {0.1469 2.5 1e-8}
POLY_OPEN_MODELING {10 0.020 50 0.5} {0.1469 2.5 1e-8}
SD_SHORT_MODELING {10 0.060 16.666 0.5} {5.000e-03 2.5 1e-8}
METAL1_SHORT_MODELING {10 0.023 43.48 0.5} {0.1469 3.0 1e-8}
METAL1_OPEN_MODELING {10 0.023 43.48 0.5} {0.1469 3.0 1e-8}
METAL2_SHORT_MODELING {10 0.028 35.71 0.5} {0.1469 3.0 1e-8}
METAL2_OPEN_MODELING {10 0.028 35.71 0.5} {0.1469 3.0 1e-8}
METAL3_SHORT_MODELING {10 0.028 35.71 0.5} {0.1469 3.0 1e-8}
METAL3_OPEN_MODELING {10 0.028 35.71 0.5} {0.1469 3.0 1e-8}
METAL4_SHORT_MODELING {10 0.044 22.73 0.5} {0.1469 3.0 1e-8}
METAL4_OPEN_MODELING {10 0.044 22.73 0.5} {0.1469 3.0 1e-8}
METAL5_SHORT_MODELING {10 0.044 22.73 0.5} {0.1469 3.0 1e-8}
METAL5_OPEN_MODELING {10 0.044 22.73 0.5} {0.1469 3.0 1e-8}
METAL6_SHORT_MODELING {10 0.044 22.73 0.5} {0.1469 3.0 1e-8}
METAL6_OPEN_MODELING {10 0.044 22.73 0.5} {0.1469 3.0 1e-8}
ILS_MODELING METAL1 METAL2 {1.0e-09}
single_OD_CONT_MODELING {0.5e-9}
single_POLY_CONT_MODELING {0.5e-9}
single_NDIFF_CONT_MODELING {0.5e-9}
single_PDIFF_CONT_MODELING {0.5e-9}
single_BUTTED_CONT_MODELING {0.5e-9}
iso_VIAx_MODELING {2 5e-1 3 5e-2 5e-3 4 5e-4 2 4e-4}
single_VIAx_MODELING {0.3e-9}
large_VIAx_MODELING {2.0 1.0e-9}
rectangle_VIA0_MODELING {0.5e-9}
iso_CONT_MODELING {1 3.18e-9}
OD_CONT_MODELING {10 0.100 10.000 0.5} {5.139e-03 2.5 1e-8}
POLY_CONT_MODELING {10 0.060 16.666 0.5} {2.366e-03 2.5 1e-8}
CAB_LI_SHORT_MODELING {10 0.150 4.167 0.5} {4.631e-03 2.5 1e-8}
CAB_LI_OPEN_MODELING {10 0.025 4.167 0.5} {1.509e-02 2.5 1e-8}
CA_LI_SHORT_MODELING {10 0.150 4.167 0.5} {4.631e-03 2.5 1e-8}
CA_LI_OPEN_MODELING {10 0.090 3.556 0.5} {4.412e-02 2.5 1e-8}
CB_LI_SHORT_MODELING {10 0.120 5.556 0.5} {4.412e-03 2.5 1e-8}
CB_LI_OPEN_MODELING {10 0.020 4.167 0.5} {1.509e-02 2.5 1e-8}
TS_LI_SHORT_MODELING {10 0.100 3.450 0.5} {4.412e-03 2.5 1e-8}
TS_LI_OPEN_MODELING {10 0.090 3.556 0.5} {4.412e-02 2.5 1e-8}
TSPC_LI_SHORT_MODELING {10 0.050 4.115 0.5} {4.255e-03 2.5 1e-8}
TSCA_LI_SHORT_MODELING {10 0.080 2.555 0.5} {1.555e-02 2.5 1e-8}
TSCB_LI_SHORT_MODELING {10 0.100 3.450 0.5} {4.412e-03 2.5 1e-8}
single_VIA0_MODELING {1e-8}
single_CA_LI_MODELING {1e-9}
single_CB_LI_MODELING {1e-9}
single_TS_LI_MODELING {1e-9}
single_NDIFF_LI_MODELING {1e-9}
single_PDIFF_LI_MODELING {1e-9}
```

[Example 11-5](#) on page 363 shows the contents of an example CAA defect file using the User-Defined List Format.

Example 11-5. Defect Data File (User-Defined List Format)

DEFECT SIZE LIST	DEFECT DENSITY LIST
DIFF_SHORT_USER_DEFINED {0.09 0.18 0.27 0.36 0.54 0.72 1.08 1.44 1.8}	{182 68.6 28.9 12.
2 5.4 1.7 0.55 0.15 0.05}	
DIFF_OPEN_USER_DEFINED {0.09 0.18 0.27 0.36 0.54 0.72 1.08 1.44 1.8}	{182 68.6 28.9 12.
2 5.4 1.7 0.55 0.15 0.05}	
POLY_SHORT_USER_DEFINED {0.09 0.18 0.27 0.36 0.54 0.72 1.08 1.44 1.8}	{182 68.6 28.9 12.
2 5.4 1.7 0.55 0.15 0.05}	
POLY_OPEN_USER_DEFINED {0.09 0.18 0.27 0.36 0.54 0.72 1.08 1.44 1.8}	{182 68.6 28.9 12.
2 5.4 1.7 0.55 0.15 0.05}	
M1_SHORT_USER_DEFINED {0.09 0.18 0.27 0.36 0.54 0.72 1.08 1.44 1.8}	{182 68.6 28.9 12.
2 5.4 1.7 0.55 0.15 0.05}	
M1_OPEN_USER_DEFINED {0.09 0.18 0.27 0.36 0.54 0.72 1.08 1.44 1.8}	{182 68.6 28.9 12.
2 5.4 1.7 0.55 0.15 0.05}	
M2_SHORT_USER_DEFINED {0.09 0.18 0.27 0.36 0.54 0.72 1.08 1.44 1.8}	{182 68.6 28.9 12.
2 5.4 1.7 0.55 0.15 0.05}	
M2_OPEN_USER_DEFINED {0.09 0.18 0.27 0.36 0.54 0.72 1.08 1.44 1.8}	{182 68.6 28.9 12.
2 5.4 1.7 0.55 0.15 0.05}	
M3_SHORT_USER_DEFINED {0.105 0.21 0.315 0.42 0.63 0.84 1.26 1.68 2.1}	{182 68.6 28.9 12.
2 5.4 1.7 0.55 0.15 0.05}	
M3_OPEN_USER_DEFINED {0.105 0.21 0.315 0.42 0.63 0.84 1.26 1.68 2.1}	{182 68.6 28.9 12.
2 5.4 1.7 0.55 0.15 0.05}	
M4_SHORT_USER_DEFINED {0.105 0.21 0.315 0.42 0.63 0.84 1.26 1.68 2.1}	{182 68.6 28.9 12.
2 5.4 1.7 0.55 0.15 0.05}	
M4_OPEN_USER_DEFINED {0.105 0.21 0.315 0.42 0.63 0.84 1.26 1.68 2.1}	{182 68.6 28.9 12.
2 5.4 1.7 0.55 0.15 0.05}	
M5_SHORT_USER_DEFINED {0.105 0.21 0.315 0.42 0.63 0.84 1.26 1.68 2.1}	{182 68.6 28.9 12.
2 5.4 1.7 0.55 0.15 0.05}	
M5_OPEN_USER_DEFINED {0.105 0.21 0.315 0.42 0.63 0.84 1.26 1.68 2.1}	{182 68.6 28.9 12.
2 5.4 1.7 0.55 0.15 0.05}	
M6_SHORT_USER_DEFINED {0.105 0.21 0.315 0.42 0.63 0.84 1.26 1.68 2.1}	{182 68.6 28.9 12.
2 5.4 1.7 0.55 0.15 0.05}	
M6_OPEN_USER_DEFINED {0.105 0.21 0.315 0.42 0.63 0.84 1.26 1.68 2.1}	{182 68.6 28.9 12.
2 5.4 1.7 0.55 0.15 0.05}	
M7_SHORT_USER_DEFINED {0.105 0.21 0.315 0.42 0.63 0.84 1.26 1.68 2.1}	{182 68.6 28.9 12.
2 5.4 1.7 0.55 0.15 0.05}	
M7_OPEN_USER_DEFINED {0.105 0.21 0.315 0.42 0.63 0.84 1.26 1.68 2.1}	{182 68.6 28.9 12.
2 5.4 1.7 0.55 0.15 0.05}	
M8_SHORT_USER_DEFINED {0.105 0.21 0.315 0.42 0.63 0.84 1.26 1.68 2.1}	{182 68.6 28.9 12.
2 5.4 1.7 0.55 0.15 0.05}	
M8_OPEN_USER_DEFINED {0.105 0.21 0.315 0.42 0.63 0.84 1.26 1.68 2.1}	{182 68.6 28.9 12.
2 5.4 1.7 0.55 0.15 0.05}	
M9_SHORT_USER_DEFINED {0.105 0.21 0.315 0.42 0.63 0.84 1.26 1.68 2.1}	{182 68.6 28.9 12.
2 5.4 1.7 0.55 0.15 0.05}	
M9_OPEN_USER_DEFINED {0.105 0.21 0.315 0.42 0.63 0.84 1.26 1.68 2.1}	{182 68.6 28.9 12.
2 5.4 1.7 0.55 0.15 0.05}	

Layer Names

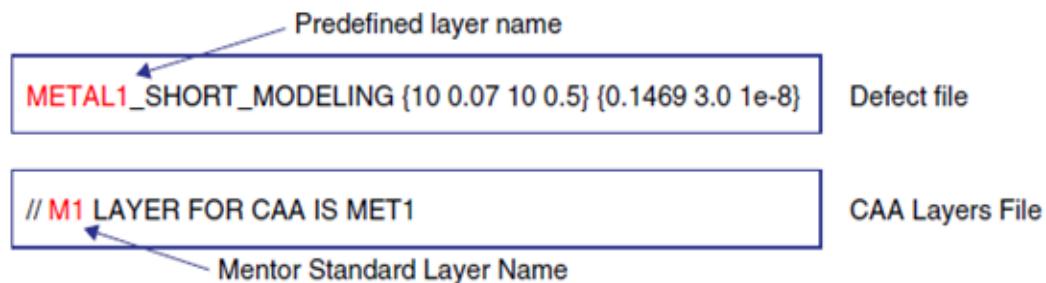
Some layer names in the file may also be predefined.

The following is a list of the predefined layer names you can use:

```
DIFF
METAL1
METAL2
METAL3
METAL4
METAL5
METAL6
METAL7
METAL8
METAL9
METAL10
METAL11
single_OD
single_POLY
single_NDIFF
single_PDIFF
```

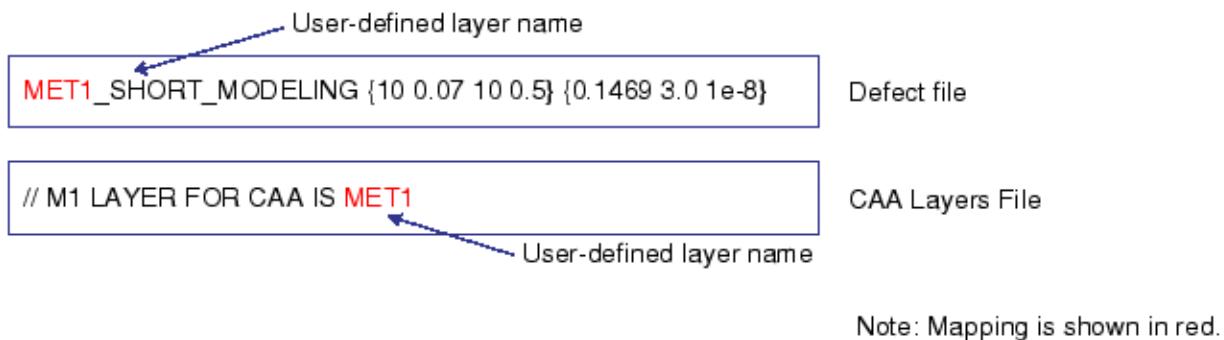
Predefined layer names map to a corresponding Mentor Standard Layer Name (refer to “[Mentor Standard Layer Names](#)” on page 407) in the “[CAA Layers File](#)” on page 366. [Figure 11-37](#) illustrates this concept.

Figure 11-37. Mapping of Predefined Layer Names



Note: Mapping is shown in red.

User-defined layer names can be specified in place of predefined names in the defect file (except for via and butted contact layers), and map directly to user-defined layer names in the CAA Layers File (see [Figure 11-38](#)).

Figure 11-38. Mapping of User-defined Layer Names

For example, the following line uses a custom name for the second metal layer:

```
CustomMetal2_SHORT_MODELING {10 0.23 10 0.5} {0.1469 3.0 1e-8}
```

You must map each user-defined name that you define in the defect file to the same user-defined name in the “[CAA Layers File](#)” on page 366. For example, assume that the CustomMetal2 layer maps to layer number 50. In this case, the CAA Layers File would contain the following lines:

```
LAYER CustomMetal2 50
...
// M2 LAYER FOR CAA IS CustomMetal2
```

Certain derived layers are generated automatically by the tool (DIFF, NDIFF and PDIFF). When predefined names are used for these layers in the defect file, no corresponding entries are needed in the CAA Layers File. However, if you wish to use custom names for these layers, they must be explicitly mapped in the CAA Layers File following the same procedure outlined above.

CAA Layers File

A CAA layers file is an ASCII file defining the layers to be used for critical area analysis.

Note

 The “_”, “#”, “-”, and “:” characters are forbidden in the *Customer_layer* as referenced in “*MGC_layer* LAYER FOR CAA IS *Customer_layer*”.

Format.....	366
Custom OD Layer Names	369

Format

There are two parts to a CAA layers file: layer definitions and layer name map.

- **Layer Definitions** — This part of the file contains standard SVRF statements and operations used to read in layer data from the layout database and derive additional layers as needed.
- **Layer Name Map** — This part of the file contains special commands used by the Calibre Critical Area Analysis tool to translate the layer names you use in the layer definition section into the names recognized by the analysis tool. Mapping is only required for the layers used by CAA. See “[Mentor Standard Layer Names](#)” on page 407 for a list of these layers.

This section can also contain marker layers (see “[CAA Analysis with Marker Layers](#)” on page 105) and exclusion layers (see “[CAA Analysis with Exclusion Layers](#)” on page 106).

Note

 Calibre performs only basic syntax checking of the CAA Layers file. It is the user’s responsibility to verify that both the metal stack and the layer mapping is correct for the chosen foundry and technology.

The layer name map section of the file is written in the form of SVRF comments, with the first three characters being two forward slashes and one space “// ”. The remaining characters must be:

MGC_layer* LAYER FOR CAA IS *Customer_layer

where:

- ***MGC_layer*** — The Mentor Graphics Standard layer name. See “[Mentor Standard Layer Names](#)” on page 407 for a list of valid names.
- ***Customer_layer*** — The name by which the layer is referenced in the layer definitions section.

Tip

 The layer map section must not contain any extra spaces or mistyped words. For best results, copy and paste out of the sample file that follows and edit the customer layer names at the end of each line.

Example 11-6. Sample CAA Layers File

```
//////START LAYER ASSIGNMENT AND DERIVATION/////
LAYER MAP 6 DATATYPE 1 10
LAYER MAP 6 DATATYPE 2 10
LAYER DIFFi 10
LAYER MAP 16 DATATYPE 1 20
LAYER MAP 16 DATATYPE 2 20
LAYER PO 20
LAYER MAP 31 DATATYPE 1 30
LAYER MAP 31 DATATYPE 2 30
LAYER CONT 30
LAYER MAP 101 DATATYPE 1 31
LAYER MAP 101 DATATYPE 2 31
LAYER ME1 31
LAYER MAP 201 DATATYPE 1 32
LAYER MAP 201 DATATYPE 2 32
LAYER ME2 32
LAYER MAP 301 DATATYPE 1 33
LAYER MAP 301 DATATYPE 2 33
LAYER ME3 33
LAYER MAP 401 DATATYPE 1 34
LAYER MAP 401 DATATYPE 2 34
LAYER ME4 34
LAYER MAP 501 DATATYPE 1 35
LAYER MAP 501 DATATYPE 2 35
LAYER ME5 35
LAYER MAP 601 DATATYPE 1 36
LAYER MAP 601 DATATYPE 2 36
LAYER ME6 36
LAYER MAP 701 DATATYPE 1 37
LAYER MAP 701 DATATYPE 2 37
LAYER ME7 37
LAYER MAP 801 DATATYPE 1 38
LAYER MAP 801 DATATYPE 2 38
LAYER ME8 38
LAYER MAP 901 DATATYPE 1 39
LAYER MAP 901 DATATYPE 2 39
LAYER ME9 39
LAYER MAP 151 DATATYPE 1 51
LAYER MAP 151 DATATYPE 2 51
LAYER VI1 51
LAYER MAP 251 DATATYPE 1 52
LAYER MAP 251 DATATYPE 2 52
LAYER VI2 52
LAYER MAP 351 DATATYPE 1 53
LAYER MAP 351 DATATYPE 2 53
LAYER VI3 53
LAYER MAP 451 DATATYPE 1 54
LAYER MAP 451 DATATYPE 2 54
LAYER VI4 54
LAYER MAP 551 DATATYPE 1 55
LAYER MAP 551 DATATYPE 2 55
LAYER VI5 55
LAYER MAP 651 DATATYPE 1 56
LAYER MAP 651 DATATYPE 2 56
LAYER VI6 56
```

```
LAYER MAP 751 DATATYPE 1 57
LAYER MAP 751 DATATYPE 2 57
LAYER VI7 57
LAYER MAP 851 DATATYPE 1 58
LAYER MAP 851 DATATYPE 2 58
LAYER VI8 58

LAYER MAP 6 DATATYPE 2 1000
LAYER MAP 6 DATATYPE 3 1001

LAYER b1 1000
LAYER b2 1001
LAYER b3 1002

LAYOUT BASE LAYER DIFFi PO CONT

/////USER PLEASE CONSTRUCT MAPPING BELOW FOR CAA/////
/////DO NOT USE " ", "#", ":" IN CUSTOMER NAMED LAYERS/////
/////BEGIN MAPPING CUSTOMER LAYERS TO MGC_CAA LAYERS/////
// OD LAYER FOR CAA IS DIFFi
// POLY1 LAYER FOR CAA IS PO
// CONT LAYER FOR CAA IS CONT
// M1 LAYER FOR CAA IS ME1
// M2 LAYER FOR CAA IS ME2
// M3 LAYER FOR CAA IS ME3
// M4 LAYER FOR CAA IS ME4
// M5 LAYER FOR CAA IS ME5
// M6 LAYER FOR CAA IS ME6
// M7 LAYER FOR CAA IS ME7
// M8 LAYER FOR CAA IS ME8
// M9 LAYER FOR CAA IS ME9
// VIA1 LAYER FOR CAA IS VI1
// VIA2 LAYER FOR CAA IS VI2
// VIA3 LAYER FOR CAA IS VI3
// VIA4 LAYER FOR CAA IS VI4
// VIA5 LAYER FOR CAA IS VI5
// VIA6 LAYER FOR CAA IS VI6
// VIA7 LAYER FOR CAA IS VI7
// VIA8 LAYER FOR CAA IS VI8
// EXCLUDE LAYER FOR CAA IS b1
// EXCLUDE LAYER FOR CAA IS b2
// EXCLUDE LAYER FOR CAA IS b3
/////END OF MAPPING CUSTOMER LAYER TO MGC_CAA LAYERS/////
```

Custom OD Layer Names

A custom name can be specified for the OD layer in the CAA layers file. If a custom name is specified, then this name is used as the rule name in Calibre RVE for DFM.

Specify the custom name in both the CAA layers file and the CAA defect file. In the following example, the custom name is “FINFET”:

- CAA Layer File:

```
FINFET = DIFFdrawn AND FINdrawn  
// OD LAYER FOR CAA IS FINFET
```

- MGC Defect File:

```
FINFET_SHORT_MODELING {10 0.032 62.500 0.5} {6.076e-05 2.5 1e-8}  
FINFET_OPEN_MODELING {10 0.020 100.000 0.5} {3.001e-05 2.5 1e-8}
```

Upon completion of the analysis, Calibre RVE for DFM displays the rule names as follows:

- FINFET.OPEN
- FINFET.SHORT

If the custom name is not specified in the CAA defect file, then the standard Mentor name applies. See “[Mentor Standard Layer Names](#)” on page 407.

Memory Configuration File Format

If your design contains embedded SRAMs with repair resources consisting of redundant rows or columns, you need to prepare a memory configuration file (*memconfig.txt*) that specifies the available repair resources.

This file is read by the Calibre Critical Area Analysis tool and incorporated into the TVF rule file. The resulting reports have Lambda and yield calculated with and without redundancy for comparison.

The memory configuration file consists of two sections. The first section, which may be only one line, defines a configuration of layers and defect types that are mitigated by column and row redundancy. Each individual memory type in the design (1024x16, 256x63, ...) is represented by a memory cell specification line that references one of the configuration definitions. The memory cell specification lines give the actual configuration of the memory in terms of column and row numbers. If all memory types in the design use the same configuration definition, then all memory cell specification lines would reference that configuration.

The format of the memory configuration file is:

```
# comment line  
config_line  
spec_line  
...
```

where *config_line* is specified as:

```
config_name '=' '{' '{' column_layer [fraction] '}' ... '}' '{' '{' row_layer [fraction] '}' ... '}'
```

- **config_name** — Specifies the name of the configuration block.
- **column_layer** — Specifies the name of the column layer or rule. Note that CAA layer names are specified as *layer_name.SHORT* or *layer_name.OPEN*, and cut layers are specified as *single.layer_name*.
- **row_layer** — Specifies the name of the row layer or rule. Note that CAA layer names are specified as *layer_name.SHORT* or *layer_name.OPEN*, and cut layers are specified as *single.layer_name*.
- **fraction** — Specifies the failures for a particular layer and defect type; applies to either column or row failures. The sum of fractions for a given layer and defect type must be less than or equal to 1. If a layer name appears in both column and row lists without a fraction following it, the default is assumed to be 0.5 for each.

and where *spec_line* is specified as:

```
memory_block config_name total_columns redundant_columns total_rows redundant_rows  
dummy_columns dummy_rows [bitcell_name]
```

- **memory_block** — Name of the memory block.
- **config_name** — Name of the configuration block specified in the first section of the file.
- **total_columns** — Specifies the total number of columns.
- **redundant_columns** — Specifies the number of redundant columns.
- **total_rows** — Specifies the total number of rows.
- **redundant_rows** — Specifies the number of redundant rows.
- **dummy_columns** — Specifies the number of dummy columns.
- **dummy_rows** — Specifies the number of dummy rows.
- **bitcell_name** — If a bitcell name is specified, then the effect of redundancy is applied only to the area within the memory block covered by that bitcell. If no bitcell name is specified, the redundancy is applied to the entire area of the memory block.

Note that for the contact layer (for example, CO), two layers are derived to account for diffusion and poly contacts. These are referenced as “single.ODCO” and “single.POCO”.

Example 11-7. Sample Memory Configuration File

```
# The column and row lists are always column first, then row:  
# The config line must be one line:  
  
sramConfig = { {DIFF} {single.ODCO} {M1.SHORT} {M1.OPEN} {single.VIA1i}  
    {M2.SHORT 0.6} {M2.OPEN 0.6} } { {PO.SHORT} {PO.OPEN} {single.POCO}  
    {M1.SHORT} {M1.OPEN} {single.VIA1} {M2.SHORT 0.4} {M2.OPEN 0.4}  
    {single.VIA2} {M3.SHORT} {M3_active.OPEN} }  
  
# Column and row totals must include redundant and dummy columns and rows  
# Most memory cell specification lines have only column redundancy:  
  
sram1rw_128x22_4 sramConfig 24 2 128 0 0 0 8t_core_bit  
sram1rw_256x22_4 sramConfig 24 2 256 0 0 0 8t_core_bit  
sram1rwlr_128x8_4 sramConfig 10 2 128 0 0 0 6t_core_bit  
  
# Note the use of dummy row and column before bitcell name:  
sram1rwlr_1024x32_8 sramConfig 35 2 1025 0 1 1 6t_core_bit
```

CAA Library Flow Cell Orientations

The Critical Area Analysis library flow allows you to run the analysis on a 1x1, 1x3 or 3x3 grid for each standard cell.

You can define custom orientations for the analysis grid in a sunset or in Calibre Interactive DFM. The default orientation for a 1x3 grid is N:N:N; for a 3x3 grid it is S:S:S:N:N:N:S:S:S (illustrated in [Figure 11-39](#)).

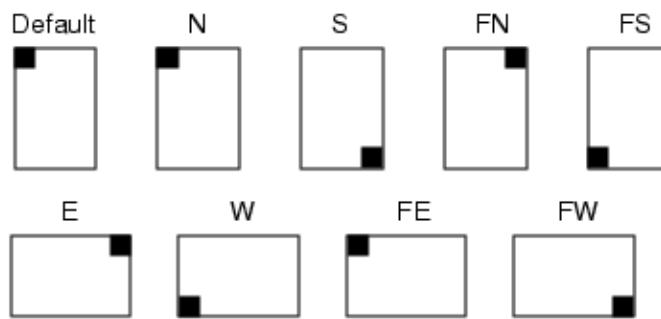
Figure 11-39. Calibre Interactive DFM Library Flow Setup



You can define custom orientations by specifying values from the following sets:

- N, S, FN, FS
- E, W, FE, FW

[Figure 11-40](#) illustrates how these values correspond to cell orientations.

Figure 11-40. Cell Orientations

All values must be specified from a single set (<{N, S, FN, FS} or {E, W, FE, FW}). In the runset file, these values are specified using the dfmLibrarySelMode and dfmLibraryPlacementGrid values. For example:

```
*dfmLibrarySelMode: 3x3
*dfmLibraryPlacementGrid: S:S:S:N:N:N:S:S:S
```

CAA What-if Analysis Configuration File Format

The What-if analysis tool has a specific configuration file format.

Refer to “[Performing CAA What-If Analysis](#)” on page 115 for more information.

The format of the what-if configuration file is as follows:

```
# comment
rule_name percentage_change
...
Die Width width_value
Die Height height_value
```

For example:

```
DIFF.OPEN          10.0
DIFF.SHORT         10.0
metall1.OPEN        -5.0
metall1.SHORT       15.0

Die Width 2
Die Height 3
```

When running the batch reporting flow, the [AnalyzeWhatIfConfigurationFile](#) key name can be used to specify a what-if configuration file. See “[DFM HTML Reporting](#)” in the *Calibre RVE User's Manual* for more information.

Fill Layermap File

A Fill layermap file is an ASCII file defining the mapping between the fill layers in a DFM database and the backannotation layers in the original layout.

For use when using backannotation from a DFM database.

Format (Fill Layermap) [374](#)

Format (Fill Layermap)

There are two parts to a Fill layermap file: comment text and layer name map.

- **Comment Text** — Lines beginning with “#” indicate comments. It is good practice to begin a Fill layermap file with two comments that serve as a header for the file.

```
#dfmdb_name layout_name <version> <date>  
#dfmdblayerNameframeworklayerName
```

- **Layer Name Map** — This part of the file contains one line for each fill layer. The line contains two layer names, both of which are case sensitive:
 - The name of the layer in the DFM database.
 - The name of the layer in the original layout.

```
m1fill    metal11  
m2fill    metal2
```

Note

 Using a Fill layermap file is optional. It is necessary when the layer names in a DFM database differ from the layer names in the original layout. When you use a layermap file, only layers listed in the file are backannotated to the databases.

DFM Analysis Reference

The table provides a summary of the CAA-specific reference topics.

Table 11-9. CAA-Specific Reference Topics

Topic	Description
Calibre Critical Area Analysis Tool	Graphical user interface for generating CAA rule decks and reports.
DFM Report Card Interface	Graphical user interface within Calibre RVE for DFM for reviewing scores and errors generated by a critical area and recommended rule analysis.
Mentor Standard Layer Names	Layer naming convention that must be used when generating a CAA rule deck using the Calibre Critical Area Analysis Tool .

Calibre Critical Area Analysis Tool

The CAA tool can generate a critical area analysis rule file, generate a defect data file, and run a set of critical area analyses using different defect data.

Accessing the CAA Tool	376
Calibre CAA Tutorial and Example Kit	383

Accessing the CAA Tool

At the command prompt, enter the command to launch Calibre Interactive DFM.

For example:

```
calibre -gui -dfm
```

In the Inputs pane, for Run, choose Critical Area Analysis (CAA). Click the **CAA** tab. [Table 11-10](#) on page 377 contains descriptions of the options available in the **Settings**, **Analysis**, **Base Layer Options**, **Wafer Analysis**, **Cells**, and **Layers** tabs.

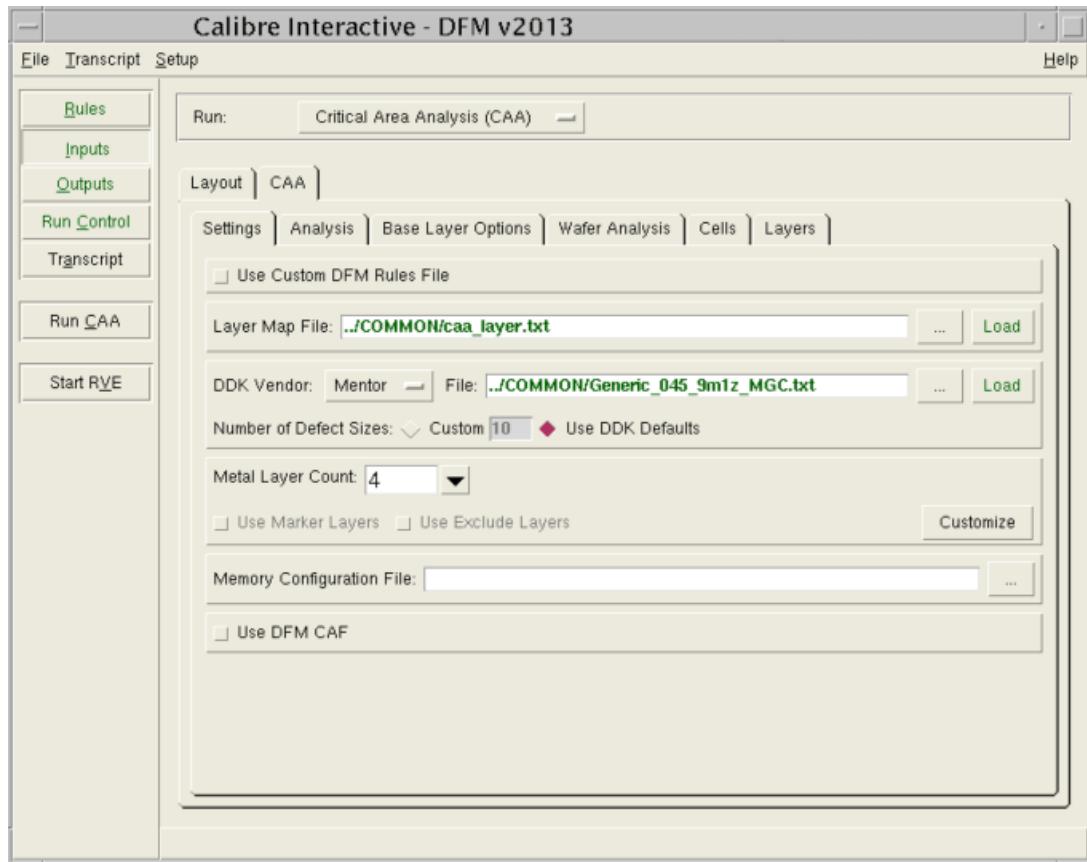


Table 11-10. Contents of Critical Area Analysis Tool

Control	Description	Recommendation
Settings Tab		
Use Custom DFM Rules File	Allows you to specify a custom DFM rule file in the Rules pane and bypass all options in the Calibre Critical Area Analysis tool.	
Layer Map File	Path to the CAA Layers File mapping technology layer names to Mentor standard layer names.	See “ Sample CAA Layers File ” on page 368.
DDK Vendor	Selects the defect input type. UMC has encrypted format. The Mentor format may or may not be encrypted.	
File	Path to the defect file.	
Number of Defect Sizes	The number of defect sizes read from the DDK Vendor file.	
pfx File, pfx Password	Path to a valid pfx file, and specification of the pfx password. Applicable to UMC defect input only.	
Metal Layer Count	Allows you to specify how many layers to use in the analysis.	Use the Customize button to specifically select or deselect individual layers.
Use Marker Layers	Enables use of marker layers. Refer to “ CAA Analysis with Marker Layers ” on page 105.	
Use Exclude Layers	Enables use of exclusion layers. Refer to “ CAA Analysis with Exclusion Layers ” on page 106.	
Memory Configuration File	Path to an optional memory configuration file. Refer to “ CAA Analysis with Memory Redundancy ” on page 108. This path is not enabled for the “ CAA NDD Flow ” on page 117.	
Use DFM CAF	Performs the analysis using the DFM CAF operation.	

Table 11-10. Contents of Critical Area Analysis Tool (cont.)

Control	Description	Recommendation
Analysis Tab		
Yield Model	<p>Specifies the formula used to calculate yield inside DFM Analyze. Lambda is calculated automatically.</p> <p>The yield models are not enabled for the “CAA NDD Flow” on page 117.</p>	Poisson is the only yield model certified by most foundries.
Model: Average Quality Metric	Specifies the formula that is used for CAA DFM scoring, not for yield prediction. Used only for the “CAA NDD Flow” on page 117.	Only enabled when defect density data is specified as “NA” in the defect file for the CAA NDD flow.
Analysis	<p>Specify the types of critical area to locate on wires:</p> <ul style="list-style-type: none"> • SHORT • OPENUWIRE • OPENWIRE 	SHORT and OPENWIRE.
Exclude Gate Edge	When analyzing for shorts, lets you exclude gate edges from the analysis.	
CA Overlap	When enabled, critical areas where a large enough defect could cause both an open and a short are analyzed and counted only once. When disabled, open and short critical areas are analyzed independently.	The net result in CA Overlap mode is a slightly higher yield prediction (approximately 2-5% on full chip runs).
Short Depth	The minimum overlap depth you believe causes a short.	

Table 11-10. Contents of Critical Area Analysis Tool (cont.)

Control	Description	Recommendation
Use METRIC OCTAGONAL	<p>Enables the METRIC OCTAGONAL option for DFM CAF and sets the metric_octagonal variable in the TVF rule deck. In the runset file, this option is saved as “*dfmOctagonalMetric”.</p> <p>DFM CAF supports this octagonal metric only for short analysis.</p> <p>Selecting this option also sets the TVF Rule deck variable “metric_square” which controls the METRIC SQUARE option (the default) for DFM CAF.</p>	<p>Use for a layout that has 45 degree and rectilinear shapes.</p> <p>This option is only available when Use DFM CAF is selected in the Settings tab.</p>
Open Depth	The maximum depth of material you believe must remain if current is to flow properly.	
Via Analysis:	<ul style="list-style-type: none"> • SINGLE • ELECTRICAL • OPEN <p>Specify the type of critical area to locate for vias: Single, Electrical, or Open.</p> <p>The Single option is a geometric check. The Single option multiplies the single via count by the single via failure rate to calculate the Lambda value.</p> <p>The Electrical option detects loops in cases where the geometric check would identify a via as single.</p> <p>The Open option checks a via shape for opens. The Open option uses an approach similar to the OPENWIRE + OPENVIA analysis for metal layers.</p>	Most foundries provide the single via failure rate, allowing you to use Single or Electrical Via.
Via Depth	The size of material that must be specified when using the OPEN option. This is the maximum depth of material you believe must remain if current is to flow properly.	

Table 11-10. Contents of Critical Area Analysis Tool (cont.)

Control	Description	Recommendation
Inter Layer Short	Enables inter-layer short analysis. When selected, this option checks for defects between layers. Defect density is checked, not the defect size. Inter-layer shorts are detected between layers that have been selected in the Layers tab.	Use with By Net analysis.
Use REGION OCTAGONAL	Enables the REGION OCTAGONAL option for DFM Critical Area and sets the region_octagonal variable in the TVF rule deck. In the runset file, this option is saved as “*dfmOctagonalRegion”. Selecting this option also sets the TVF Rule deck variable “metric_square” which controls the METRIC SQUARE option for DFM Critical Area.	Use for a layout that has 45 degree and rectilinear shapes. This option is only available when Use DFM CAF is <i>not</i> selected in the Settings tab.
Partition	Specifies the types of partitioning to use for analysis and reporting. You can select By Chip or By Library, By Window (only with By Chip), By Cell or By Net. For By Library, refer to “ CAA Library Flow Cell Orientations ” on page 372 for further details on defining cell orientations. Refer to “ Partitioning ” on page 259.	If you select By Window, you must also specify a WINDOW Size .
Win Size	Defines the WINDOW Size when Partitioning by Window .	The default window size is 50 um.
By Cell	Enables By Cell partitioning to use for analysis and reporting. For By Cell and By Net, specify items in include or exclude using the Specify List button.	

Table 11-10. Contents of Critical Area Analysis Tool (cont.)

Control	Description	Recommendation
By Net	Enables By Net partitioning to use for analysis and reporting. You can specify a list of nets to include using the Specify List button. Note that By Net is enabled only when the Use DFM CAF option is enabled in the Settings tab.	
CA Output	Instructs the tool to write results to the DFM database and defines which results to output: <ul style="list-style-type: none"> • None • All • First n 	Use None to save runtime and generate reports without writing results. Note that None is required when using DFM CAF.
Large Size Threshold	A scaling factor (in terms of minCD) defining a threshold for extracting critical area. Above this threshold, the application shifts to fast mode.	
Die Area	An optional field used to normalize results by die area. The calculated yield models are divided by this value and converted to the appropriate units, which allows you to report metrics per unit area.	Useful for comparing designs or cells of different sizes.
Extract CA Mask	Allows you to override the EXTRACT_CA_MASK layer declaration in the layer map file.	
Base Layer Options Tab		
Extraction Mode	Allows you to change base layer options via a dropdown menu.	Modify base layer options based on your foundry requirements.

Table 11-10. Contents of Critical Area Analysis Tool (cont.)

Control	Description	Recommendation
Wafer Analysis Tab		
Gross/Good Die Per Wafer	<p>Enables calculation of Gross Die Per Wafer or Good Die Per Wafer (GDPW). Gross Die Per Wafer is the total number of whole die that fit on the wafer. Good Die Per Wafer is the product of CAA Yield and Gross Die Per Wafer.</p> <p>This calculation option is not enabled for the “CAA NDD Flow” on page 117.</p>	
Custom Manual	<p>Selects the flow to use:</p> <ul style="list-style-type: none"> The Custom flow allows you to manually enter values for Wafer Diameter, Edge Width, Street Width, and Number of Test Dies. The Manual flow allows you to specify the Number of Gross Dies per wafer. When using this option, specify an integer larger than 0. 	
Flat <ul style="list-style-type: none"> • Yes • No 	Specifies whether or not the wafer contains a flat edge on one side. The default is No.	
Cells Tab		
Cell Usage File	<p>Specifies a file containing typical usage rates for a list of cells. These values replace the Instance Count in Calibre RVE for DFM, and produce different Lambda and Yield values for flat metrics.</p> <p>The format for each line of the cell usage file is: <i>cell_name cell_count</i></p> <p>The option is enabled only when By Library is selected from the Analysis tab.</p>	
Selected Cells	Restricts the analysis to the specified cells.	

Table 11-10. Contents of Critical Area Analysis Tool (cont.)

Control	Description	Recommendation
Exclude Cells from Reports	Excludes the specified cells from reporting, but includes them in total calculations.	
Exclude Cells	Excludes the specified cells from the analysis.	
Layers Tab		
Use	Allows you to choose specific layers to include in the analysis.	
Controls Pane		
Run CAA	Click to run the analysis.	
Start RVE	Loads Calibre RVE for DFM with the generated DFM database.	

Calibre CAA Tutorial and Example Kit

A tutorial and example kit (eKit) is available to you that demonstrates using the CAA tool.

Try It! 	Calibre Critical Area Analysis Tutorial and Example Kit Includes documentation and data for performing Critical Area Analysis (CAA) with Calibre YieldAnalyzer. The procedures step you through running CAA on a chip, CAA with marker layers, CAA on a cell library, and CAA with No Defect Density (NDD). Go to this page on Support Center to download the eKit (Documentation tab, Document Types=Getting Started Guide). The link goes to the latest release.
---	---

DFM Report Card Interface

Use this tool to review scores and errors generated by a critical area or recommended rule analysis.

Accessing the DFM Report Card Interface..... 384

Accessing the DFM Report Card Interface

Follow the steps to access the DFM Report Card.

1. Load your layout database into your layout viewer.
2. From the layout viewer, invoke Calibre RVE for DFM and load the DFM database containing the results and scores from your analysis.

In Calibre RVE for DFM, the DFM report card is visible when you first invoke the tool.

Note

 The DFM Report Card is functional only if your database is a DFM Database containing the required annotations, as described in the *Calibre YieldServer Reference Manual*.

Also, the ability to highlight data in your layout viewer from within the DFM Report Card is only available if your layout is loaded into the layout viewer and socket communication is set up between the layout viewer and Calibre RVE for DFM. You set up socket communication automatically when you launch the tool from your layout viewer.

Calibre RVE for DFM Report Card

This section describes the Calibre RVE for DFM report card interface that you see when you load a DFM database.

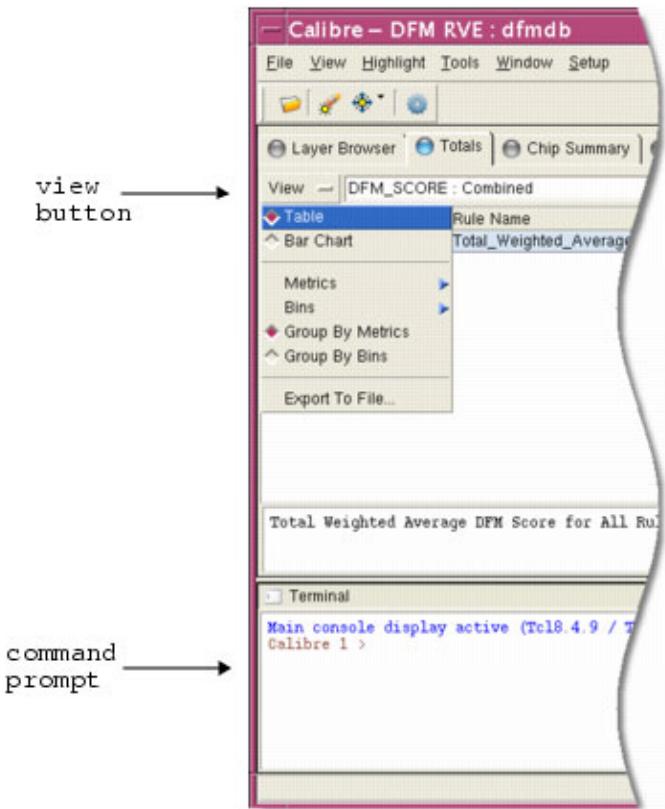
Overview	385
Filtering Data by Flat Metrics	388
Tab Contents	389

Overview

The Calibre RVE for DFM report card is an interface that displays DFM check and score data in a tabular format.

You can control the data displayed through a set of special-purpose annotations. In order for DFM results to display properly, the rule writer must insert commands that generate the required annotations. See “[Writing Rules That Display in DFM Report Card](#)” on page 59.

Note that Calibre RVE for DFM is invoked automatically with the -turbo option enabled (MultiThreaded mode) when it is loaded from a layout viewer (such as Calibre DESIGNrev). When Calibre RVE for DFM is invoked from Calibre Interactive DFM, the options in the Run Control section determine whether or not -turbo is enabled. You can set the environment variable MGC_RVE_DISABLE_TURBO to 1 to disable the -turbo option when Calibre RVE for DFM is invoked from a layout viewer or from Calibre Interactive DFM.



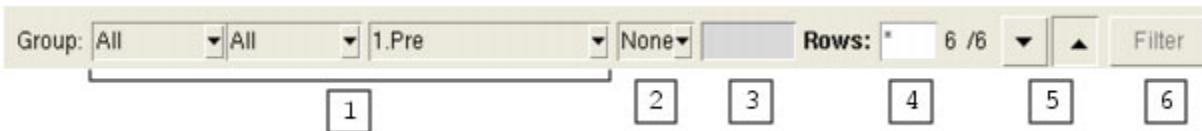
The **View** button is located in the upper left corner on all tabs except the **Layers** tab. When clicked, it displays a dropdown list of the following options:

- **Table | Bar Chart** — Toggles the data field between table display (the default) and bar chart display.
- **Metrics** — Selects the metrics to display in the data field. By default, all metrics are visible.
- **Bins** — Selects the bins to display in the data field. By default, all bins are visible.
- **Group by Metrics | Group by Bins** — Toggles column grouping. When **Group by Metrics** (the default) is selected, columns are grouped such that all bins are displayed for the first metric, then all bins for the second metric, and so on. When **Group by Bins** is selected, columns are grouped such that all metrics for the first bin are displayed, then all metrics for the second bin, and so on.
- **Export to File** — Writes the current displayed data to either a tab-delimited (Report) or comma-delimited (CSV) file.

The command prompt provides a method of running Calibre YieldServer commands interactively. It can be enabled by choosing **Setup > Options > Session** and selecting **Terminal Pane**. Click **Apply**, then **Close**.

Filtering Data

- All tabs except the **Layers** tab contain filtering controls that allow you to control how score data is displayed. By default, scores less than or equal to zero are not shown.



- Area 1** — Select the group, priority (applies only to the **Chip Summary** tab), or rule (applies only to the **Chip Summary**, **Cell Summary**, and **Window Summary** tabs) and the bin and metric combination to apply filtering to.

For priority, specify “All”, “None”, or any of the defined priorities. “None” selects rules with empty priorities. “All” selects all priorities for all rules.

- Area 2** — Choose the threshold operator to apply on the selected bin and metric combination. Operator choices are: greater than (>), greater than or equal to (>=), less than (<), less than or equal to (<=), or equal to (==).
- Area 3** — Enter the threshold value to apply on the selected group and the bin and metric combination. Enter a blank value to apply no filtering.
- Area 4** — Enter the number of result entries to show among those that pass the filter criteria (enter “*” to display all).
- Area 5** — Choose whether to display results in ascending or descending order.
- Area 6** — Click the **Filter** button to apply the currently displayed filter settings.

Note that none of these options, when changed, automatically update the display. You must click the **Filter** button to apply any changes you have made.

For example, to show the two highest scores on the entire chip:

- Select the **Chip Summary** tab.
- Select “All” groups, “All” priorities, and the “metric: Combined” value in area 1.
- Choose “>=” in area 2.
- Enter “0” in area 3.
- Enter “2” in area 4.
- Click the “up” button in area 5 to sort in descending order.
- Click **Filter**.

Filtering Data by Flat Metrics

You can sort by flat metrics from the **Cell Summary** tab.

Procedure

1. Enable display of flat metrics by choosing **View > Show Flat Metrics**.
2. Choose **View > Filter By Flat Metrics**.

When this option is enabled, the filter controls apply to metrics that appear in the flat columns. When this option is disabled (the default), the filter controls apply to the metrics that appear in the hierarchical columns.

3. Click **Filter** to update the sorting in the table.

Tab Contents

The following sections describe the tab contents.

Totals Tab.....	389
Chip Summary Tab.....	391
Cell Summary Tab.....	394
Window Summary Tab.....	396
Drill Down Tab.....	398
Layer Browser Tab	400
Highlighting Errors on a Specific Layer	402
Recreating and Highlighting Unsaved Derived Layers in a Layout Viewer	403
Drilling Down into Problem Spots on a Layer	403
Setting the Colormap Display Options	404
Choosing Metrics and Bins to Display.....	404
Finding Hot Spots in a Design	405
Analyzing Score Data by Cell.....	405
Changing Histogram and Colormap Colors	406

Totals Tab

This tab shows all rules with the “DFM_TYPE” annotation set to “total”.

One use of this tab is to display the total weighted average DFM score for all rules.

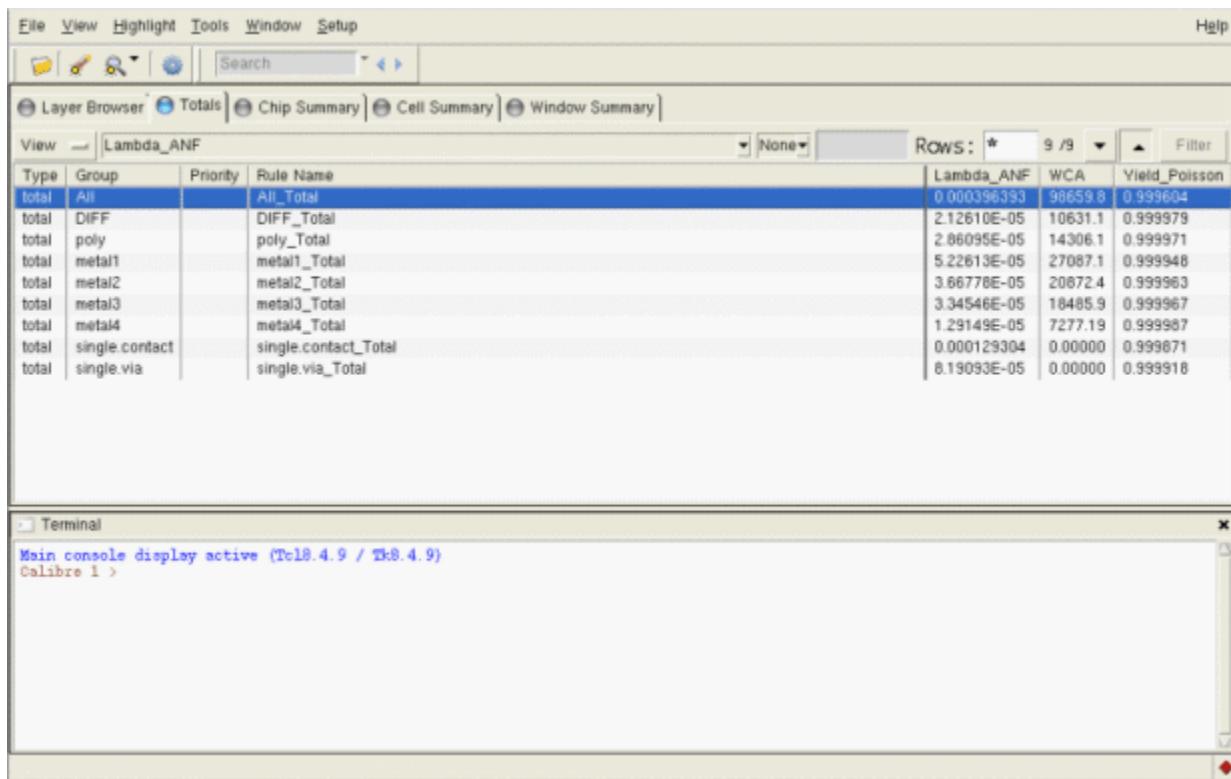


Table 11-11. Contents of the Totals Tab

Name	Description
Column Headers	
Type	Displays rules with the “DFM_TYPE” annotation set to “total”.
Group	One or more groups that a rule belongs to, displayed as “group1 group2 / ...”. Specified with one or more instances of the “DFM_GROUP” annotation.
Priority	Rule priority. Specified with the “DFM_PRIORITY” annotation.
Rule Name	Name of the rule. Specified with the “DFM_RULE” annotation.
metric: Combined	Displays the sum of scores across all bins, per metric, for each group.
metric: b0...bn	Displays individual bin scores, per metric, for each group.

Table 11-11. Contents of the Totals Tab (cont.)

Name	Description
Right Click Menu Options	
Cell Summary	Highlights the Cell Summary tab and displays scores for the selected rule by cell.
Window Summary	Highlights the Window Summary tab and displays scores for the selected rule by window.
Browse Hierarchy Errors	Generates a separate Calibre RVE for DFM window that allows you to browse errors hierarchically for the selected rule.
Browse Flat Errors	Generates a separate Calibre RVE for DFM window that allows you to browse errors flat for the selected rule.
Waive Errors	Waives the error reported by a rule check.
Highlight Hierarchy Errors	Highlights errors hierarchically for the selected rule in the layout viewer.
Highlight Flat Errors	Highlights errors flat for the selected rule in the layout viewer.
Histogram Hierarchy Cells	Generates a histogram that displays the cells in which results for the selected rule appear versus score severity.
Colormap Hierarchy Cells	Generates a color map that displays cells in which results for the selected rule appear. A layout viewer does not need to be running for this option to work.
Histogram Hierarchy Windows	Generates a histogram that displays the windows in which results for the selected rule appear versus score severity.
Colormap Hierarchy Windows	Generates a color map that displays windows in which results for the selected rule appear. A layout viewer does not need to be running for this option to work.
Histogram Flat Errors	Generates a histogram that displays the flat error count versus score severity for the selected rule.
Colormap Flat Errors	Generates a full color map of flat errors for the selected rule in Calibre RVE for DFM.

Chip Summary Tab

Use this tab to view score data for the full chip.

Options you can access in **Setup > Options > DFM Browser** allow you to control how data is displayed for this tab. Refer to “[Setup DFM Browser Options Pane](#)” in the *Calibre RVE User’s Manual* for a list of settings.

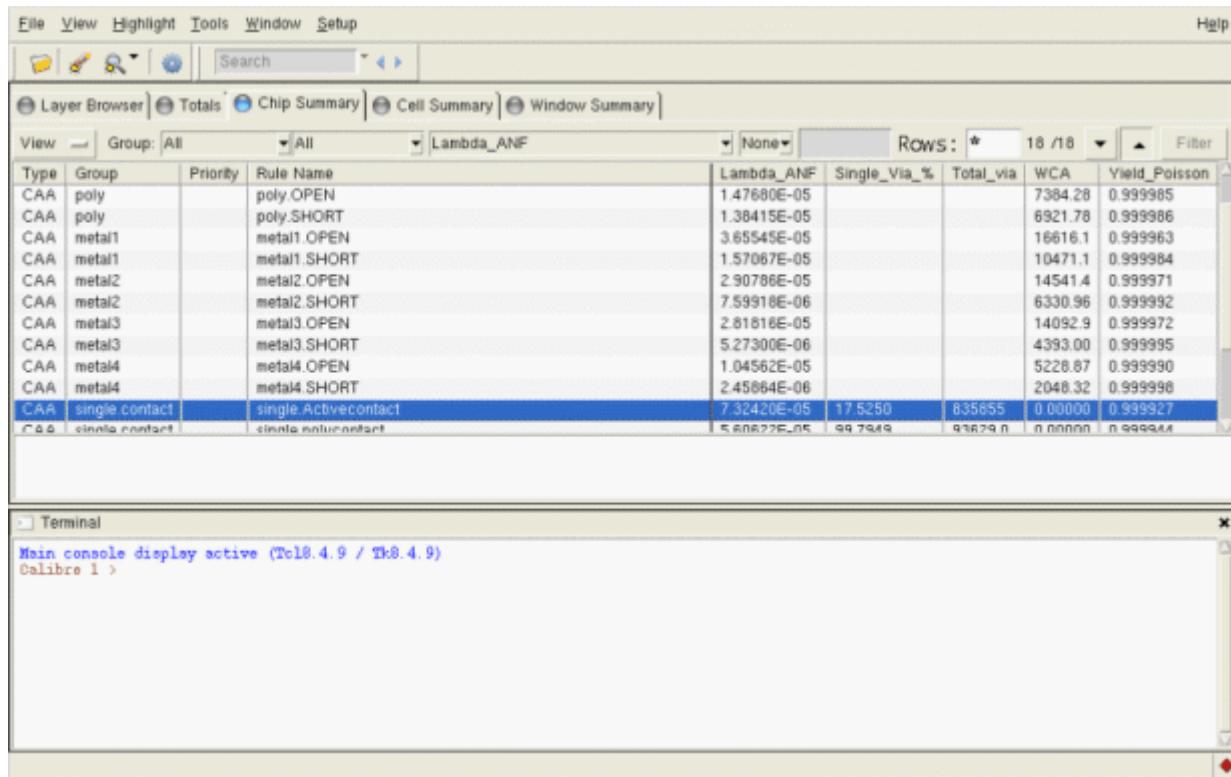


Table 11-12. Contents of the Chip Summary Tab

Name	Description
Column Headers	
Type	Rule type. Specified with the “DFM_TYPE” annotation
Group	One or more groups that a rule belongs to, displayed as “group1 group2 / ...”. Specified with one or more instances of the “DFM_GROUP” annotation.
Priority	Rule priority. Specified with the “DFM_PRIORITY” annotation.
Rule Name	Name of the rule. Specified with the “DFM_RULE” annotation.
<i>metric:</i> Combined	Displays the sum of scores across all bins, per <i>metric</i> , for each rule.

Table 11-12. Contents of the Chip Summary Tab (cont.)

Name	Description
<i>metric:</i> b0...bn	Displays individual bin scores, per <i>metric</i> , for each rule.
Right Click Menu Options	
Cell Summary	Highlights the Cell Summary tab and displays scores for the selected rule by cell.
Window Summary	Highlights the Window Summary tab and displays scores for the selected rule by window.
Browse Hierarchy Errors	Generates a separate Calibre RVE for DFM window that allows you to browse errors hierarchically for the selected rule.
Browse Flat Errors	Generates a separate Calibre RVE for DFM window that allows you to browse errors flat for the selected rule.
Waive Errors	Waives the error reported by a rule check.
Highlight Hierarchy Errors	Highlights errors hierarchically for the selected rule in the layout viewer.
Highlight Flat Errors	Highlights errors flat for the selected rule in the layout viewer.
Histogram Hierarchy Cells	Generates a histogram that displays the cells in which results for the selected rule appear versus score severity.
Colormap Hierarchy Cells	Generates a color map that displays cells in which results for the selected rule appear. A layout viewer does not need to be running for this option to work.
Histogram Hierarchy Windows	Generates a histogram that displays the windows in which results for the selected rule appear versus score severity.
Colormap Hierarchy Windows	Generates a color map that displays windows in which results for the selected rule appear. A layout viewer does not need to be running for this option to work.
Histogram Flat Errors	Generates a histogram that displays the hierarchical error count versus score severity for the selected rule.
Colormap Flat Errors	Generates a full color map of flat errors for the selected rule in Calibre RVE for DFM.

Cell Summary Tab

Use this tab to view score data by cell.

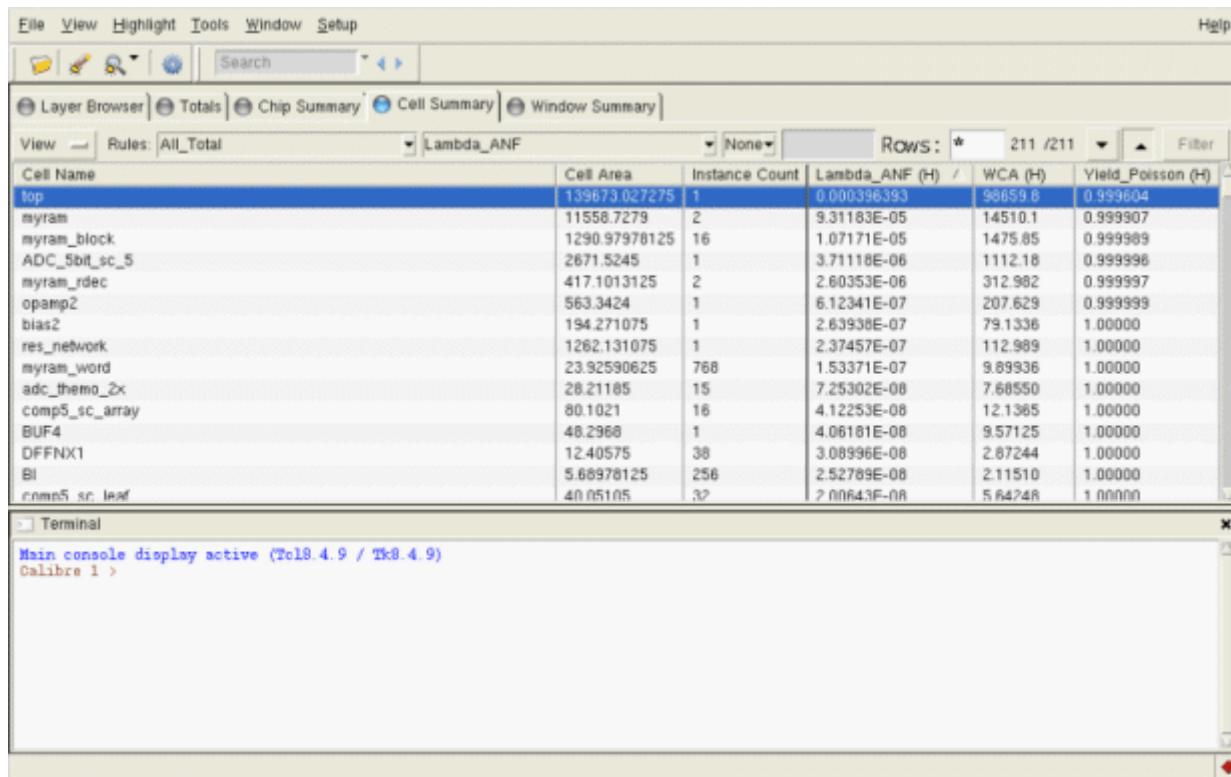


Table 11-13. Contents of the Cell Summary Tab

Name	Description
Column Headers	
Cell Name	Name of the cell.
Instance Count	The number of instances of the cell in the design.
<i>metric:</i> Combined	Displays the sum of scores across all bins, per <i>metric</i> , for each cell.
<i>metric:</i> b0...bn	Displays individual bin scores, per <i>metric</i> , for each cell.
Right Click Menu Options	
Drill Down	Highlights the Drill Down tab and displays the individual rules that comprise the total score for the selected cell.

Table 11-13. Contents of the Cell Summary Tab (cont.)

Name	Description
Browse Hierarchy Errors	Generates a separate Calibre RVE for DFM window that allows you to browse errors hierarchically for the selected cell.
Highlight Cell	Highlights the selected cell in the layout viewer.
Highlight Hierarchy Errors	Highlights all errors in the selected cell in the layout viewer.
Histogram Hierarchy Errors	Generates a histogram that displays the error count versus score severity for the selected cell.
Colormap Hierarchy Errors	Generates a color map of errors for the selected cell in the layout viewer.
Histogram Hierarchy Cells	Generates a histogram that displays the cells in which results for the selected rule appear versus score severity.
Colormap Hierarchy Cells	Generates a color map that displays cells in which results for the selected rule appear. A layout viewer does not need to be running for this option to work.

Window Summary Tab

Use this tab to view score data by window.

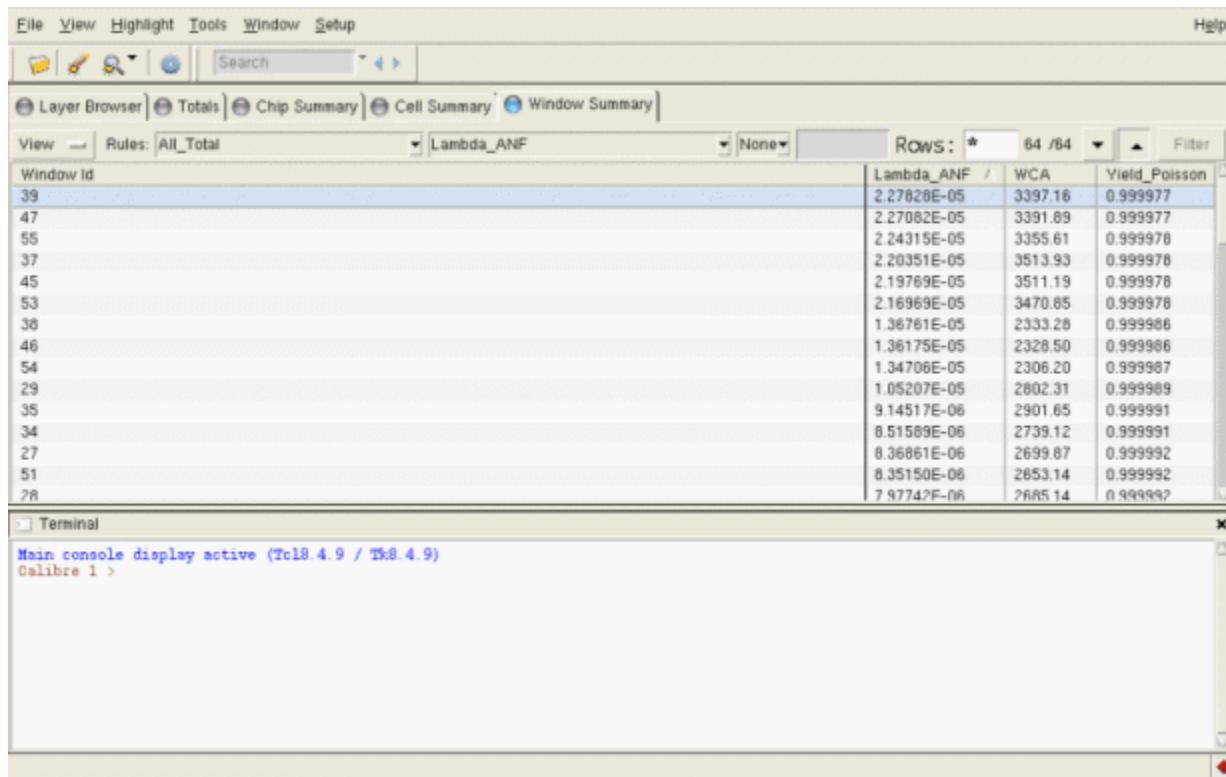


Table 11-14. Contents of the Window Summary Tab

Name	Description
Column Headers	
Window Id	Window identification number.
Coordinates	Coordinates for a window as it appears on the design. Coordinate pairs are displayed for the lower-left and upper right corners of the window.
<i>metric:</i> Combined	Displays the sum of scores across all bins, per <i>metric</i> , for each window.
<i>metric:</i> b0...bn	Displays individual bin scores, per <i>metric</i> , for each window.
Right Click Menu Options	
Drill Down	Highlights the Drill Down tab and displays the individual rules that comprise the total score for the selected window.

Table 11-14. Contents of the Window Summary Tab (cont.)

Name	Description
Browse Hierarchy Errors	Generates a separate Calibre RVE for DFM window that allows you to browse errors for the selected window.
Highlight Window	Highlights the selected window in the layout viewer.
Highlight Hierarchy Errors	Highlights all errors in the selected window in the layout viewer.
Histogram Flat Errors	Generates a histogram that displays the flat error count versus score severity for the selected window.
Colormap Hierarchy Errors	Generates a color map of errors for the selected window in the layout viewer.
Histogram Hierarchy Windows	Generates a histogram that displays the windows in which results for the selected rule appear versus score severity.
Colormap Hierarchy Windows	Generates a color map that displays windows in which results for the selected rule appear. A layout viewer does not need to be running for this option to work.

Drill Down Tab

Use this tab to view score data associated with a particular cell or window.

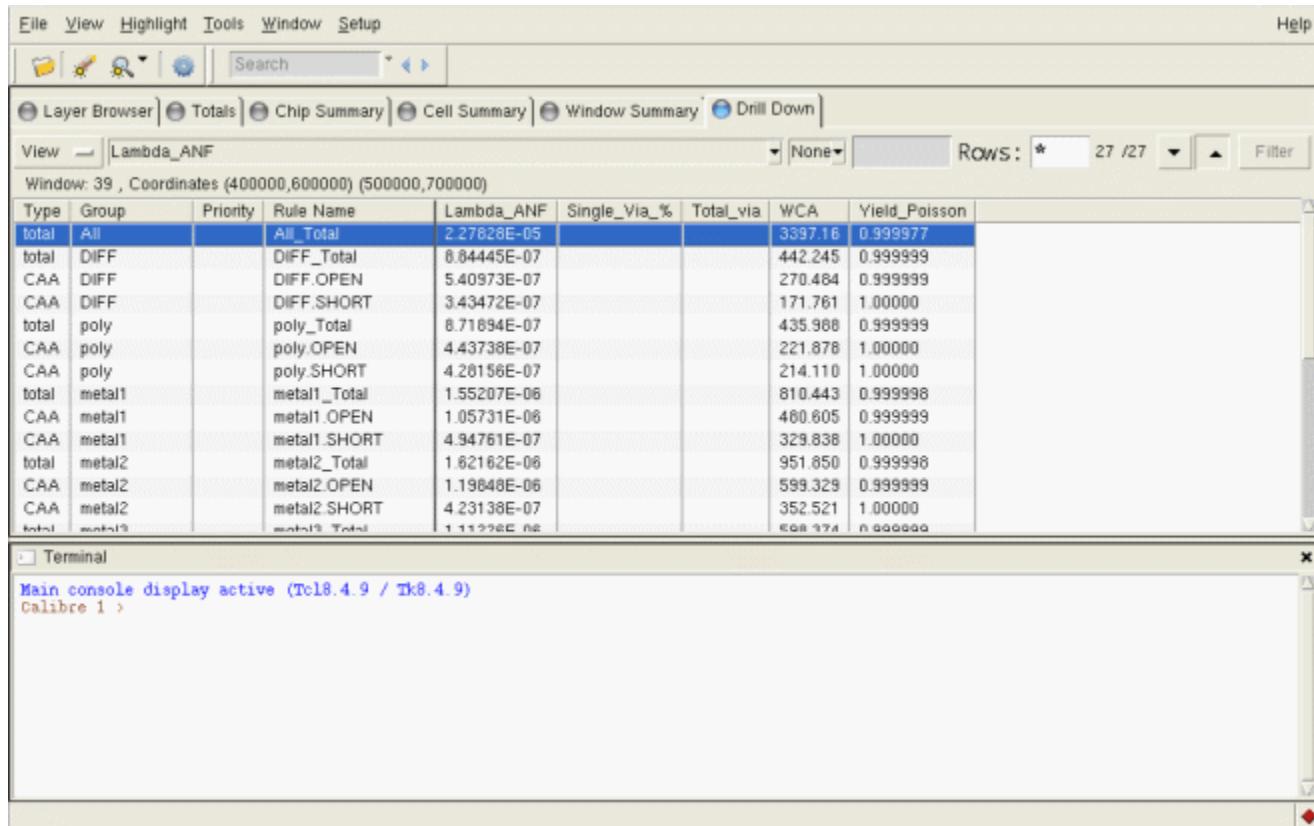


Table 11-15. Contents of the Drill Down Tab

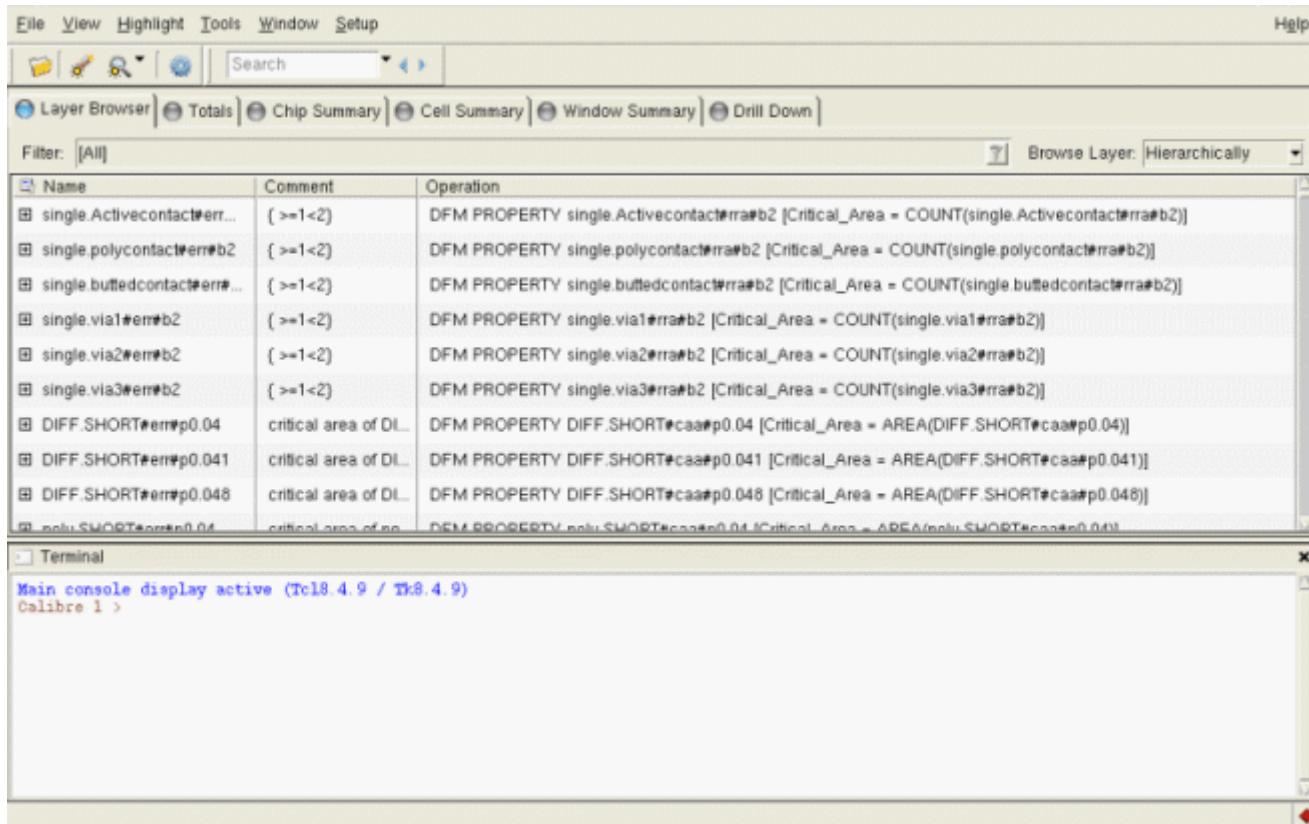
Name	Description
Column Headers	
Type	Rule type. Specified with the “DFM_TYPE” annotation.
Group	One or more groups that a rule belongs to, displayed as “group1 group2 / ...”. Specified with one or more instances of the “DFM_GROUP” annotation.
Priority	Rule priority. Specified with the “DFM_PRIORITY” annotation.
Rule Name	Name of the rule.
<i>metric:</i> Combined (H)	Displays the sum of scores across all bins, per <i>metric</i> , for each rule.

Table 11-15. Contents of the Drill Down Tab (cont.)

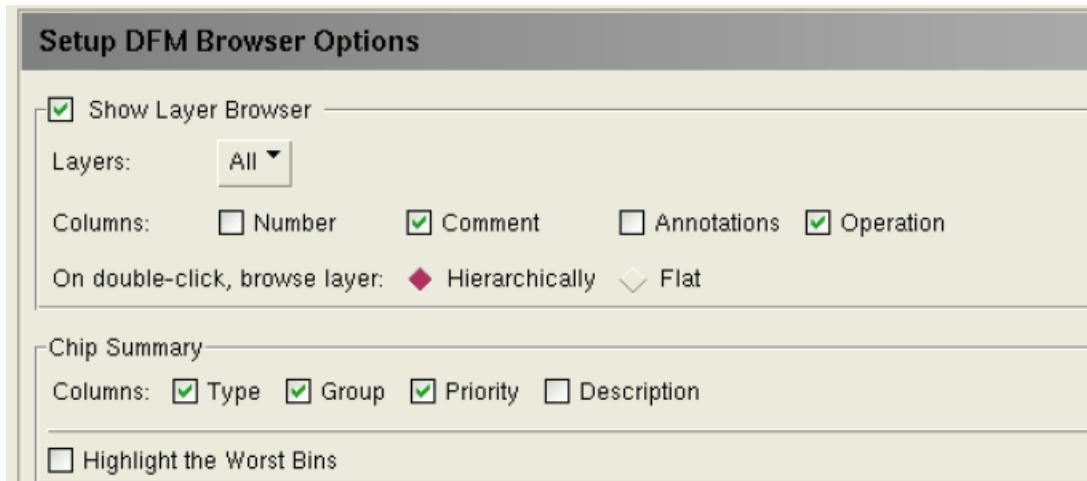
Name	Description
<i>metric:</i> b0...bn (H)	Displays individual bin scores, per <i>metric</i> , for each rule.
Right Click Menu Options	
Browse Errors	Generates a separate Calibre RVE for DFM window that allows you to browse errors in a particular cell or window.
Highlight Errors	Highlights all errors for the selected rule in a particular cell or window.
Histogram Errors	Generates a histogram that displays the error count versus score severity for a particular cell or window.
Colormap Errors	Generates a color map of errors for the selected rule in a particular cell or window in Calibre RVE for DFM.
Highlight Critical Area	Highlights critical area shapes in a cell. This is valid for library analysis. The highlighting works for individual rules that correspond to Chip Summary rules, not Total Summary items.
Show Cell Highlight Window	Highlights a particular cell or window on the layout.

Layer Browser Tab

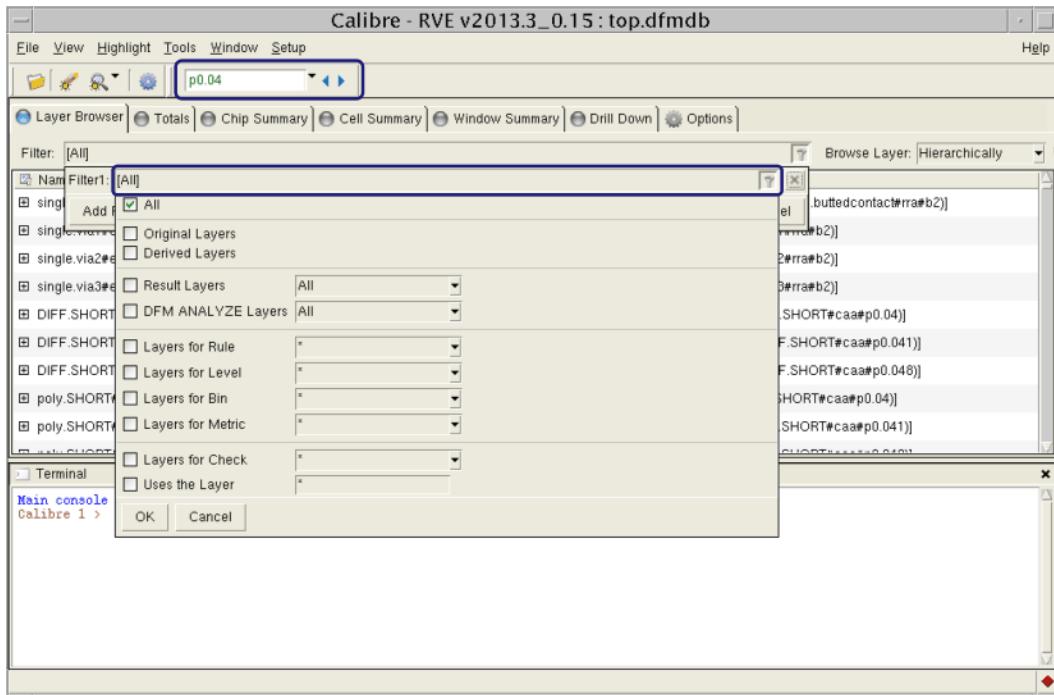
Use this tab to examine errors on a layer-by-layer basis or examine a specific area on the selected layer.



The Layer Browser tab displays a table of data read from your DFM database. Each table row contains a different layer and is divided into columns that show the layer number, its name, a comment (if one exists) and an expandable tree containing the operation(s) that derived it. Options you can access in **Setup > Options > DFM Browser** enable you to control how layer data is displayed. Refer to “[Setup DFM Browser Options Pane](#)” in the *Calibre RVE User’s Manual* for a list of settings.



You can use the **Search** and **Filter** features to quickly locate layers of interest in the **Layer Browser** tab:



To find a layer, enter a string in the text field next to Find and press the icon. You can scroll through previous occurrences of the string using the icon. To match the case of the string or use regular expressions, choose Match Case or Regex from the dropdown menu, respectively. The results are returned for strings in any visible column (starting with the left-most one).

The **Filter** menu allows you to show or hide specific types of layers. Its options are described in [Table 11-16](#).

Table 11-16. Layer Browser Filter Menu Options

Filter Name	Description
All	Displays all layers.
Original Layers	Displays only original layers.
Derived Layers	Displays only derived layers.
Result Layers	Displays only result layers (in most cases, these layers are created with a DFM operation).
DFM Analyze Layers	Displays only layers created with the DFM Analyze operation.
Layers for Specific Rule	Displays all layers for which the DFM_RULE annotation exists and the value of the annotation matches the string specified in the text field. Wildcards (*) are supported.
Layers for Specific Level	Displays all layers for which the DFM_LEVEL annotation exists and the value of the annotation matches the string specified in the text field. Wildcards (*) are supported.
Layers for Specific Bin	Displays all layers for which the DFM_BIN annotation exists and the value of the annotation matches the string specified in the text field. Wildcards (*) are supported.
Layers for Specific Metric	Displays all layers for which the DFM_METRIC annotation exists and the value of the annotation (which is normally a list of metrics) matches the string specified in the text field. Wildcards (*) are supported.
Layers for Check	Displays all layers that are the result of the check.
Uses the Layer	Display all layers that were created using the specified layer. Wildcards (*) are supported.

To apply the filter, choose the desired settings and click **OK**. The layer list updates to match the chosen settings.

Highlighting Errors on a Specific Layer

You can highlight errors on a specific layer using Calibre RVE for DFM.

Procedure

1. Choose Browse Layer Hierarchically or Flat.

2. Right-click and select Browse (or double click on the layer) to see either hierarchical results only, or all results after they are promoted to the top level, respectively. Calibre RVE for DFM opens the standard Calibre RVE window, displaying the errors or capture windows from the selected window.
3. Highlight results from the standard Calibre RVE window using one of the typical methods:
 - Double-click a result.
 - Right-click a result for a context-sensitive highlight menu (all applications).

The selected error or errors are highlighted in the layout viewer.

Recreating and Highlighting Unsaved Derived Layers in a Layout Viewer

You can use Calibre RVE for DFM to recreate and highlight unsaved derived layers.

Note that the derived layer's original layers and any necessary connection layers must have been saved in the DFM database, otherwise the tool issues an error message.

From the **Layer Browser** tab's display of the tree, you can see the sequence of derivations that resulted in the final derived layer. Depending on your SVRF rule file, some of these derived layers may not have been saved in the DFM database. You can, however, recreate and highlight these unsaved derived layers as follows:

Procedure

1. Ensure you have Calibre RVE for DFM connected to your layout viewer.
2. From the Calibre RVE for DFM **Layer Browser** tab, use the left-mouse button to select the layer you want to highlight.
3. With the layer selected, click and hold the right-mouse button and select **Highlight Hierarchy Layer** from the popup.

Drilling Down into Problem Spots on a Layer

Layer drill down is a useful method to locate problems on a layer.

Use the following steps to drill down:

Procedure

1. Choose **View > Create Layer**
2. Type valid SVRF code into the text area. You may also copy and paste SVRF code directly from the layers table by highlighting a layer and selecting the **Copy to Create** right-click menu option, or choosing **View > Copy to Create**.

3. Click **Create**.

Calibre RVE for DFM takes the text you typed and uses it as the input to dfm::new_layer. You are essentially running individual SVRF commands interactively from within Calibre RVE for DFM. Note that any layers you create using the **Create** button exist only in memory and are lost when you exit Calibre RVE for DFM.

Setting the Colormap Display Options

Color map display options provide a method to visualize the DFM results.

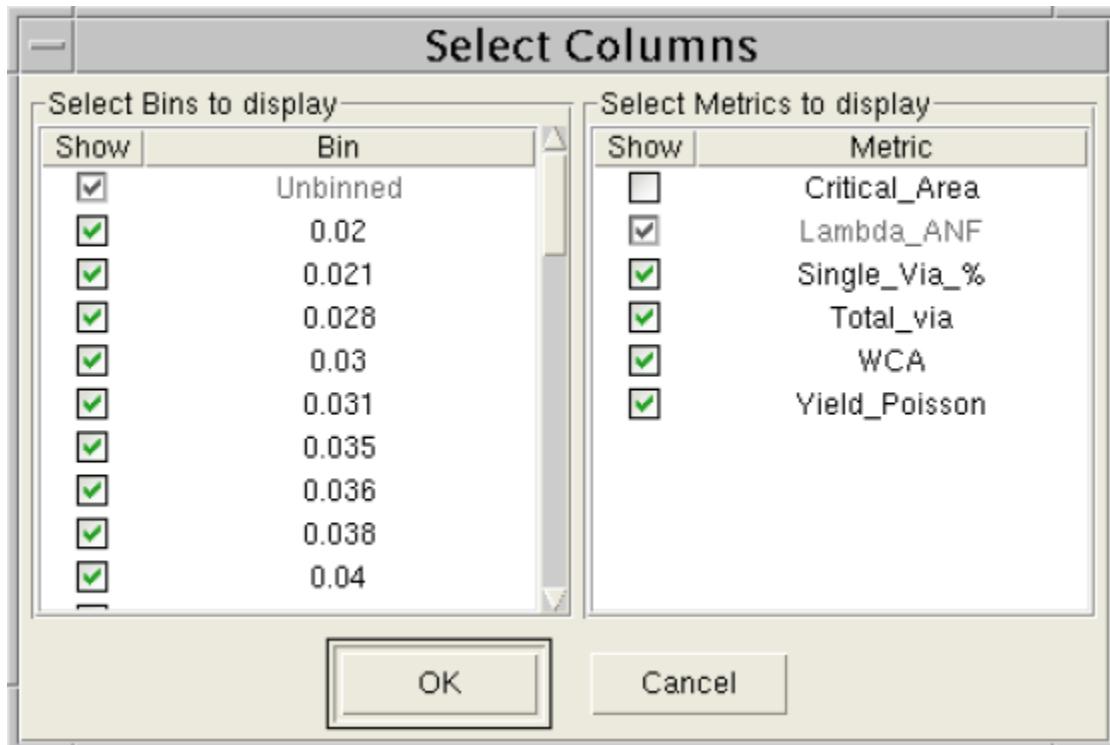
Procedure

1. In Calibre RVE for DFM, choose **Setup > Options**.
2. Click the **Histograms & Colormaps** category.
3. In the Colormaps (DFM Browser only) section of the pane, click the Show Borders checkbox to display colormaps with band borders.
4. Click **Apply**, then **Close**.

Choosing Metrics and Bins to Display

From the **View** menu, select **Metrics > Select Multiple**.

Choose the combination of bins and metrics that you wish to display and click **OK**.



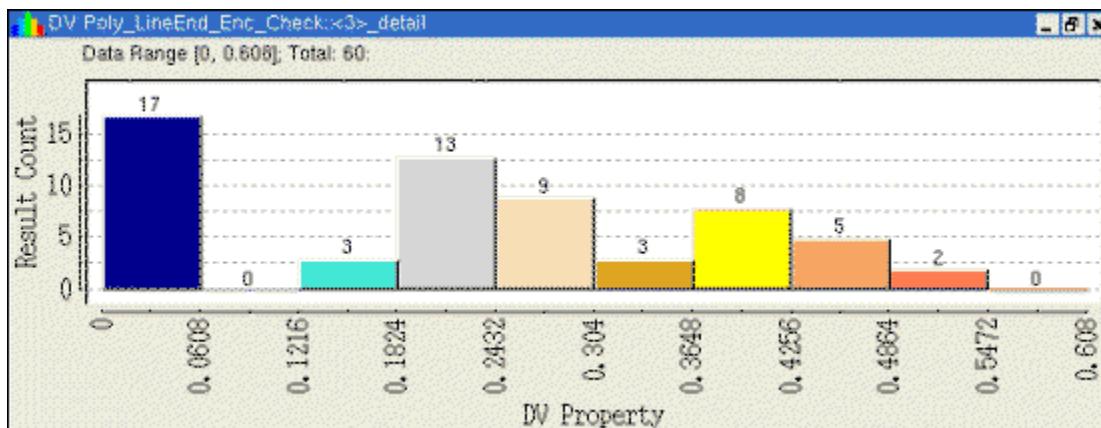
Finding Hot Spots in a Design

You can generate histograms of result counts versus scores to identify areas in your design most likely to affect yield.

Analyzing Score Data by Cell

From the **Chip Summary** tab, right click on a rule and select **Histogram Hierarchy Cells > DFM_SCORE: Combined**.

A histogram displays in the lower half of the Calibre RVE for DFM window. In this case, the number of cells (60) is shown, ordered by score severity, for the Cont_Gate_Space rule.



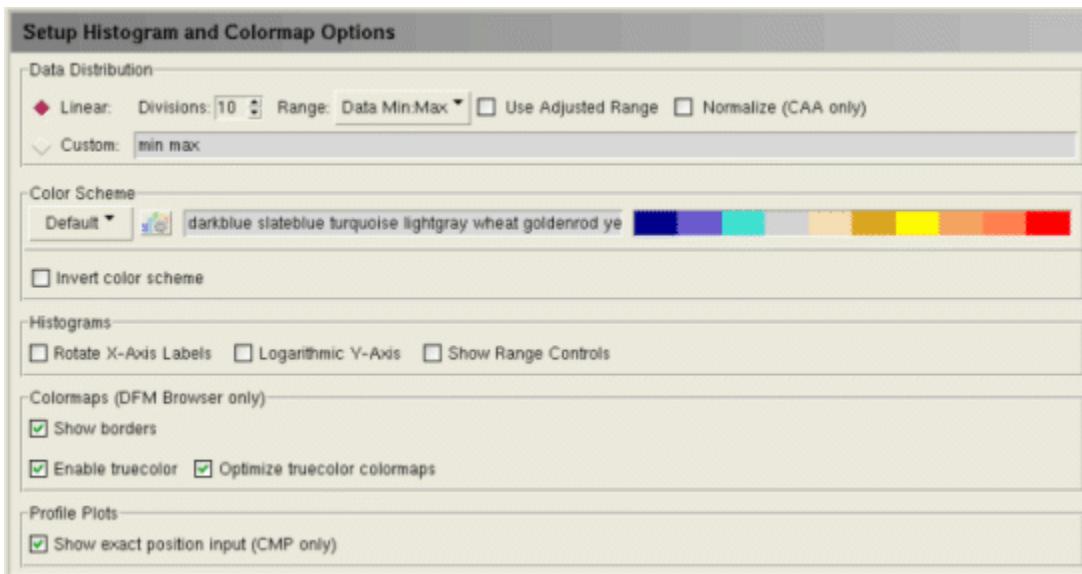
- To display a full color map of the cells in which rules in each score range appear, right click in the histogram window and select Show Color Map. This option requires a layout viewer to be running.
- To display a color map of the cells in the design where results in a particular range appear, right click on the histogram bar in the desired range and select Highlight This Bar. This option requires a layout viewer to be running.

Changing Histogram and Colormap Colors

You can change the global color gradient used in histograms in colormaps through the **Setup > Options > Histograms** menu.

Under Color Scheme, choose Custom Colors to define a new color scheme. Each new color is separated by a space, and can be a text name or hexadecimal code. Use the palette button to

enter hexadecimal codes directly from the palette. Click **Apply** then **Close** to apply the new color gradient.



Color settings are saved in the *.rvedb* configuration file.

Note that if a DFM_METRIC_COLORS database annotation is defined for a metric, Calibre RVE for DFM uses the color scheme defined in the annotation for that metric.

Mentor Standard Layer Names

Mentor standard layer names are layer names that the Calibre Critical Area Analysis tool recognizes.

This tool keys off these names to perform the proper type of analysis for each layer.

OD
RX
CA
CB
TS
VIA0
POLY1
CONT
M1
M2
M3
M4
M5
M6
M7
M8
M9
M10
M11
VIA1
VIA2
VIA3
VIA4
VIA5
VIA6
VIA7
VIA8
VIA9
VIA10

Appendix A

DFM Report Card

The Calibre RVE for DFM user interfaces are designed to display DFM check and scores data in a report card format.

The report card format simplifies the task of interpreting DFM results and planning how to improve the design under investigation. You control the display of data in the DFM Report Card using a set of special-purpose (reserved) annotations. See “[DFM Report Card Controls](#)” on page 410.

In order for DFM results to display properly, the rule writer must insert commands that generate the required annotations. See “[Writing Rules That Display in DFM Report Card](#)” on page 59.

DFM Report Card Controls	410
Layer Annotations.....	410
Database-Level Annotations.....	413
Example DFM Report Card	419

DFM Report Card Controls

This section describes DFM Report Card controls.

When reading from a DFM database, the DFM Report Card uses certain [DFM Annotations](#) that you provide in your rule deck inside of [DFM Analyze](#), [DFM RDB](#), or DFM Database operations. There are two types of annotations recognized by the DFM Report Card:

- [Layer Annotations](#) — Control how rule checks are interpreted and displayed in the DFM Report Card.
- [Database-Level Annotations](#) — Enable custom visualization ranges, control total calculation methods, and control how to display cell scores.

Layer Annotations	410
Database-Level Annotations	413

Layer Annotations

Layer annotations are used to report specific information in the DFM Report Card.

You can add layer annotations to your rule file using the following format and constraints:

ANNOTATE '['*name* = "string"]'

- The brackets [] are required.
- The name can be an unquoted string and need not be unique.
- The *string* must be enclosed by double quotes ("").

The supported layer annotations include the following:

- **DFM_BIN** — Required annotation that assigns the check to a bin. Bins are displayed as a set of columns in the DFM Report Card. The syntax is as follows:

DFM_BIN "string"

where:

- **string** — Any alphanumeric string. Can include the pipe (|), colon (:), dollar sign (\$), hyphen (-), percent sign (%), and semicolon (;) special characters. Whitespace is not allowed.

- **DFM_DESCRIPTION** — Optional annotation that defines text that appears at the bottom of the DFM Report Card window when the check is highlighted. Multiple DFM_DESCRIPTION annotations are allowed for the same layer. The syntax is as follows:

DFM_DESCRIPTION "string"

where:

- **string** — Any alphanumeric string. Can include the pipe (|), colon (:), dollar sign (\$), hyphen (-), percent sign (%), and semicolon (;) special characters. Whitespace is allowed.
- **DFM_GROUP** — Optional annotation that defines a group for the check, which you can use as a parameter for filtering in the DFM Report Card. May be specified more than once. The syntax is as follows:

```
DFM_GROUP "string"
```

where:

- **string** — Any alphanumeric string. Can include the pipe (|), colon (:), dollar sign (\$), hyphen (-), percent sign (%), and semicolon (;) special characters. Whitespace is allowed.
- **DFM_LAYER_PROPERTY_RANGE** — Allows you to set a custom range for layer property histograms/colormaps. The syntax is as follows:

```
DFM_LAYER_PROPERTY_RANGE = "property,range"
```

where:

- **property** — Specifies the layer property name that the custom range is applied to.
- **range** — The actual custom range value using the following format:

```
min_value max_value
```

You can specify this annotation value using a Calibre YieldServer script—see “Annotations in DFM Databases” in the *Calibre YieldServer Reference Manual*.

- **DFM_LEVEL** — Required annotation that specifies what level of analysis to perform. The syntax is as follows:

```
DFM_LEVEL "analysis_level"
```

where:

- **analysis_level** — A literal that specifies the analysis to perform. Choose one of the following:
 - “chip”
 - “cell”
 - “window”
 - “error”

- **DFM_METRIC** — Required annotation that assigns a name to the expression used to calculate scores. The syntax is as follows:

```
DFM_METRIC "string"
```

where:

- **string** — Any alphanumeric string. Can include the pipe (|), colon (:) , dollar sign (\$), hyphen (-), percent sign (%), and semicolon (;) special characters. Whitespace is allowed.

- **DFM_PRIORITY** — Optional annotation that assigns a priority to the check, which can be used as a sorting parameter in the DFM Report Card. The syntax is as follows:

```
DFM_PRIORITY "string"
```

where:

- **string** — Any alphanumeric string. Can include the pipe (|), colon (:) , dollar sign (\$), hyphen (-), percent sign (%), and semicolon (;) special characters. Whitespace is not allowed.

- **DFM_RULE** — Required annotation that specifies the name of the rule. Appears in the Rule Name column in the DFM Report Card.

```
DFM_RULE "string"
```

where:

- **string** — Any alphanumeric string. Can include the pipe (|), colon (:) , dollar sign (\$), hyphen (-), percent sign (%), and semicolon (;) special characters. Whitespace is not allowed.

- **DFM_TYPE** — Required annotation that specifies the type of analysis to perform. If this annotation is set to “total”, the rule appears in the **Totals** tab in the DFM Report Card. The syntax is as follows:

```
DFM_TYPE "string"
```

where:

- **string** — Any alphanumeric string. Can include the pipe (|), colon (:) , dollar sign (\$), hyphen (-), percent sign (%), and semicolon (;) special characters. Whitespace is allowed.

- **DFM_WAIVING** — Optional annotation that is used to prevent waiving of a specified rule:metric:bin for the error level appropriate layer. If this annotation is set to “disable”, “disabled”, or “off” (case-insensitive), the right mouse button **Waive Errors** operation in DFM Report Card and the referenced DRC RVE view (opened after a browse operation) are disabled for the specified rule:metric:bin. The syntax is as follows:

```
DFM_WAIVING "string"
```

- **DFM_WEIGHT** — Optional annotation that is used to calculate group totals when the calculation method is set to “Weighted Average.” Refer to “[Database-Level Annotations](#)” on page 413 for more information on calculation methods. The syntax is as follows:

```
DFM_WEIGHT "value"
```

where:

- **value** — A floating-point numeric value.

Database-Level Annotations

There are several database-level annotations that you can add to your rule file to enhance design analysis.

With DFM database-level annotations, you can perform the following actions:

- Enable custom visualization ranges and total calculations.
- Specify how to display flat cell scores.
- Control which bins and metrics are displayed by default.
- Customize the color gradient for colormaps and histograms.
- Specify which column in the DFM Report Card should be used for default sort order and direction of the sort.

You add database-level annotations to your rule file using the following format and constraints:

ANNOTATE [’name = “string”’]

- The brackets [] are required.
- The *name* can be an unquoted string and need not be unique.
- The *string* must be enclosed by quotation marks (“ ”).

The following lists the supported database-level annotations:

- **DFM_METRIC_RANGE** — Allows you to specify custom bin ranges for all histograms and colormaps for a particular metric. The syntax is as follows:

```
DFM_METRIC_RANGE = "metric,range,divisions,sort"
```

where:

- **metric** — Matches a DFM_METRIC annotation on one or more rules in the database.
- **range** — Whitespace-separated list of two or more values.

- ***divisions*** — The number of divisions within a range, if applicable.
- ***sort*** — Specifies the sort order. This argument must be either INVERTED (sort low-to-high) or NONINVERTED (sort high-to-low). The default is NONINVERTED.

Example A-1. DFM_METRIC_RANGE

```
ANNOTATE [DFM_METRIC_RANGE = "DFM_SCORE,0 1,5"]
// Generates 5 equally spaced divisions between 0 and 1

ANNOTATE [DFM_METRIC_RANGE = "DFM_SCORE,0.0 0.2 0.5 0.7 1.2"]
// Defines both the range and division values

ANNOTATE [DFM_METRIC_RANGE = "DFM_SCORE,min max,10"]
// Generates 10 equally spaced divisions between the
// data minimum and maximum
```

- **DFM_METRIC_TOTAL_METHOD** — Defines the method for total calculations on a per-metric basis and allows you to specify how to display cell scores. The syntax is as follows:

```
ANNOTATE [DFM_METRIC_TOTAL_METHOD =
"metric,calc_method,cell_ref_total"]
```

where:

- ***metric*** — Matches the value of a DFM_METRIC annotation on one or more rules in the database. Only checks or scores that have DFM_METRIC set to this value are included in the total scores.
- ***calc_method*** — Specifies the method used to calculate totals across rule checks. Allowed values are “Sum”, “Product”, “Average”, or “Weighted Average”. Formulas for each of these methods are listed as follows:

Sum: Total = (*check1score* + *check2score* + ... + *checknscore*)

Product: Total = (*check1score* * *check2score* * ... * *checknscore*)

Average: Total = (*check1score* + *check2score* + ... + *checknscore*) / *n*

Weighted Average: Total = (*w_1* * *check1score*) + (*w_2* * *check2score*) + ... + (*w_n* * *checknscore*)

where *n* is the number of checks factored into the score, and *w_n* is the value of the DFM_WEIGHT annotation for check *n*.

- ***cell_ref_total*** — Defines how scores are calculated for all instances of each cell. This is a second score calculated in addition to the *cell_score* (where the *cell_score* is the sum of property values associated with a rule or set of rules at the cell level for a particular metric). The *cell_ref_total* is calculated using the value of *calc_method* to combine the scores for each of the By Cell checks.

Allowed values for *cell_ref_total* are “Sum”, which uses (*cell_score* * *instance_count*), or “Product”, which uses (*cell_score* ^ *instance_count*).

Example A-2. DFM_METRIC_TOTAL_METHOD

```
DFM DATABASE "dfmdb" OVERWRITE [ALL]
ANNOTATE [DFM_METRIC_TOTAL_METHOD = "DFM_SCORE,Weighted
Average,Product"]
ANNOTATE [DFM_METRIC_TOTAL_METHOD = "SCORE2,Weighted
Average,Product"]
// Sets the DFM_SCORE and SCORE2 metrics to "Weighted Average"
total
// Calculation and "Product" cell score calculation
```

This annotation is relevant only for the analysis of the **Cell Summary** tab. Where the *calc_method* combines the effect of all the rules associated with a DFM_METRIC at the cell level (*cell_score*), the *cell_ref_total* takes the *cell_score* for all references and calculates the impact of all instances of those references. This corresponds to the flat score reported in the **Cell Summary** tab.

If the DFM_METRIC_TOTAL_METHOD annotation is specified, and there is no group that already has a calculated total, Calibre RVE for DFM calculates the total scores when the DFM database is initially loaded and creates a new revision in the DFM database. Total scores are recalculated after a critical area analysis is performed, or when **Tools > Recalculate Totals** is used.

- **DFM_BINS_TO_INCLUDE** — Specifies a comma-separated list of bins to be shown by default for a specific Report Card view. The syntax is as follows:

```
DFM_BINS_TO_INCLUDE = "analysis_level,b0,b1,...bn"
```

where:

- **analysis_level** — A value of a DFM_LEVEL annotation on one or more layers. This value defines the specific DFM Report Card view on which the default bin values are shown. The wildcard character (*) may be used to apply the setting to all levels.
- **b0,b1,...bn** — Each value *b0* to *bn* is the value of a DFM_BIN annotation on one or more layers. Single-character wildcards (?) and multi-character wildcards (*) are supported. The wildcard character may also be used to specify that all available bins should be shown by default.

Example A-3. DFM_BINS_TO_INCLUDE

```
ANNOTATE [DFM_BINS_TO_INCLUDE = "chip,0.1,0.5,1.0,2.0"]
// Specifies that for the chip level, five bins are to be shown:
// 0.1, 0.5, 1.0, 2.0 and the un-binned (usually the accumulated
// value, shown as "Combined").

ANNOTATE [DFM_BINS_TO_INCLUDE = "*",0.1,0.5,1.0,2.0]
// Specifies that for all levels, five bins are to be shown:
// 0.1, 0.5, 1.0, 2.0 and the un-binned (usually the accumulated
// value, shown as "Combined").

ANNOTATE [DFM_BINS_TO_INCLUDE = "chip,*"]
// Specifies that for the chip level, all available bins are to
// be shown.
```

- **DFM_METRICS_TO_INCLUDE** — Specifies a comma-separated list of metrics to be shown by default for a specific DFM Report Card view. The syntax is as follows:

```
DFM_METRICS_TO_INCLUDE = "analysis_level,m0,m1,...mn"
```

where:

- **analysis_level** — A value of a DFM_LEVEL annotation on one or more layers. This value defines the specific DFM Report Card view on which the metrics are shown. The wildcard character (*) may be used to apply the setting to all levels.
- **m0,m1,...mn** — Each value *m0* to *mn* is the value of a DFM_METRIC annotation on one or more layers. Single-character wildcards (?) and multi-character wildcards (*) are supported. The wildcard may also be used to specify that all available metrics should be shown by default.

For example:

```
ANNOTATE [DFM_METRICS_TO_INCLUDE =
    "chip,Critical_Area,Yield_Poisson"]
// Specifies that for the chip level, two metrics are to be shown:
// Critical_Area and Yield_Poisson.

ANNOTATE [DFM_METRICS_TO_INCLUDE = "*,_Lambda_ANF*"]
// Specifies that for all levels, all metrics that begin with
// _Lambda_ANF are to be shown.
```

- **DFM_METRIC_COLORS** — Specifies a list of colors to use for a given DFM_METRIC annotation value. This color setting applies to all histograms, DFM colormaps, and layout-colormap actions and overrides the color setting specified in the .rvedb file. The syntax is as follows:

```
DFM_METRIC_COLORS = "dfm_metric,color1,color2,...colorn"
```

where:

- **dfm_metric** — A value of a DFM_METRIC annotation. The wildcard character (*) may be used to apply the specified colors to all metrics for which a color scheme is

not already defined with a DFM_METRIC_COLORS annotation. If a metric already has a defined color scheme, it is not affected by the wildcard.

- ***color1,color2,...colorn*** — Each value *color1* to *colorn* is a color name or hexadecimal value.

Example A-4. DFM_METRIC_COLORS

```
// Specify color names to apply to the Critical_Area metric
ANNOTATE [DFM_METRIC_COLORS =
"Critical_Area,red,green,blue,white"]

// Specify colors black and white as hexadecimal values; apply to
// all undefined metrics.
ANNOTATE [DFM_METRIC_COLORS = "*,#000000,#FFFFFF"]
```

If the number of divisions in a histogram (or colors in a colormap) is greater than the number of defined colors, Calibre RVE for DFM automatically calculates the color gradient based on intermediate colors. This is useful, for example, when the gradient has defined endpoints, such as blue for the best score and red for the worst score.

If the number of divisions in a histogram (or colors in a colormap) is less than the number of defined colors, Calibre RVE for DFM drops colors from the middle of the list. For example, suppose DFM_METRIC_COLORS is defined as follows:

```
DFM_METRIC_COLORS = "m1,red,yellow,blue"
```

In this case, if a histogram has two divisions, only red and blue are shown.

- **DFM_CONTROL_CHIP_SUMMARY_COLUMNS** — Controls display of the Type, Group, and Priority columns in the **Chip Summary** and **Totals** tabs. This annotation overrides the settings specified in the *.rvedb* file in your home directory, and updates the settings in the **Chip Summary** tab of the **Options > DFM Browser** menu for the current Calibre RVE for DFM session. The syntax is as follows:

```
DFM_CONTROL_CHIP_SUMMARY_COLUMNS =
"column:showorhide[,column:showorhide ...]"
```

where:

- ***column*** — Specifies a column. Valid values are “Type”, “Group”, “Priority”, and “Description”.
- ***showorhide*** — Specifies whether to show or to hide the *column*. Valid values are “Show” and “Hide”.

Example A-5. DFM_CONTROL_CHIP_SUMMARY_COLUMNS

```
// Shows the Type and Group columns, hides the Priority column
DFM_CONTROL_CHIP_SUMMARY_COLUMNS =
"Type:Show,Group:Show,Priority:Hide,Description:Show"
```

- **DFM_SORT_KEY** — Specifies the default sort order in DFM Report Card for the CAA/CFA flow. The syntax is as follows:

```
DFM_SORT_KEY = "level, sort_key, sort_order"
```

where:

- **level** — A value on a DFM_LEVEL annotation for which the default sort order is changed. Valid values are case-insensitive and can be specified as follows:
 - “Chip” — **Chip Summary** tab data
 - “Totals” — **Totals** tab data
- **sort_key** — Specifies the column of default sort order. Can be a metric column or a meta column with the following syntax:
 - For a metric column, the syntax must match a DFM_METRIC annotation on one or more rules in the database. For example:

```
Critical_Area, Single_Via_%, Total_via
```

Note

 The Lambda_ANF, Yield_Poisson, Yield_Exp, and WCA metrics are related to the CAA flow. You can specify the default sort order for these metric columns by specifying a DFM_SORT_KEY annotation in your CAA DFM runset file with a *dfmDFMDbAnnotations statement. See [Example A-7](#).

- For a meta column, the syntax must be one of the following values: Type, Group, Priority, or Rule Name.

See [Example A-6](#).

- **sort_order** — Specifies the sort order. This case-insensitive argument must be either Ascending or Descending. The priority of this annotation is higher than DFM_METRIC_RANGE annotation priority.

Example A-6. DFM_SORT_KEY Rule Annotations

You specify the default sort order for the CFA DFM Report Card metrics and values with the following rule file statements:

```
ANNOTATE [DFM_SORT_KEY = "Chip, Critical_Area, Ascending"]
// Specifies that Chip Summary tab in DFM Report Card will be
sorted by
// Critical_Area metric column in low-to-high order
ANNOTATE [DFM_SORT_KEY = "Chip, Rule Name, Descending"]
// Specifies that Chip Summary tab in DFM Report Card will be
sorted by
// Rule Name meta column in high-to-low order
```

Example A-7. DFM_SORT_KEY Runset Annotations

You specify an ascending default sort order for the CAA DFM Report Card at the “Chip” level for the WCA metric using the following runset annotations:

```
*dfmDFMDbAnnotations: {DFM_SORT_KEY {Chip, WCA, Ascending}}
```

You specify the default sort order for the CAA DFM Report Card for multiple annotations and values at the “Chip” level and the “Totals” level for the WCA metric using the following runset annotations:

```
*dfmDFMDbAnnotations: {DFM_SORT_KEY {Chip, WCA, Descending}}  
{DFM_SORT_KEY {Totals, WCA, Ascending}}
```

Example DFM Report Card

The DFM Report Card annotations can be implemented in a rule file.

Example A-8. DFM Report Card Annotations

```

DFM DATABASE "dfmdb" OVERWRITE [ALL]
ANNOTATE [DFM_METRIC_TOTAL_METHOD = "DFM_SCORE,Sum,Product"]
ANNOTATE [DFM_METRIC_RANGE = "DFM_SCORE,0 1,5"]

Active_Enc_Gate_Check {

    DFM RDB OdEncPo_score NULL ALL CELLS
    // Output raw error shapes and properties
    ANNOTATE [DFM_METRIC = "DFM_SCORE"]
    ANNOTATE [DFM_RULE = "Active_Encl_Gate"]
    ANNOTATE [DFM_LEVEL = "error"]
    ANNOTATE [DFM_TYPE = "cfa"]
    ANNOTATE [DFM_BIN = ""]
    ANNOTATE [DFM_GROUP = "Perf"]
    ANNOTATE [DFM_PRIORITY = "2-MC"]
    ANNOTATE [DFM_DESCRIPTION = "Increase the spacing from gate poly to
inside corner of active to improve this score"]

    DFM ANALYZE OdEncPo_score sd >= 0 // output chip level statistics
    [(COUNT(sd) > 0) ? PROPERTY(OdEncPo_score,DFM_SCORE)/COUNT(sd) : 0]
    // Sum of values normalized by S/D count
    RDB ONLY DV COORD NULL
    ANNOTATE [DFM_METRIC = "DFM_SCORE"]
    ANNOTATE [DFM_RULE = "Active_Encl_Gate"]
    ANNOTATE [DFM_LEVEL = "chip"]
    ANNOTATE [DFM_TYPE = "cfa"]
    ANNOTATE [DFM_BIN = ""]
    ANNOTATE [DFM_GROUP = "Perf"]
    ANNOTATE [DFM_PRIORITY = "2-MC"]
    ANNOTATE [DFM_DESCRIPTION = "Increase the spacing from gate poly to
inside corner of active to improve this score"]

    DFM ANALYZE OdEncPo_score sd >= 0 BY CELL NOPSEUDO
    // Output cell level statistics
    [(COUNT(sd) > 0) ? PROPERTY(OdEncPo_score,DFM_SCORE)/COUNT(sd) : 0]
    // Sum of values normalized by S/D count
    RDB ONLY DV COORD NULL
    ANNOTATE [DFM_METRIC = "DFM_SCORE"]
    ANNOTATE [DFM_RULE = "Active_Encl_Gate"]
    ANNOTATE [DFM_LEVEL = "cell"]
    ANNOTATE [DFM_TYPE = "cfa"]
    ANNOTATE [DFM_BIN = ""]
    ANNOTATE [DFM_GROUP = "Perf"]
    ANNOTATE [DFM_PRIORITY = "2-MC"]
    ANNOTATE [DFM_DESCRIPTION = "Increase the spacing from gate poly to
inside corner of active to improve this score"]

    DFM ANALYZE OdEncPo_score sd >= 0 NUMWIN 40 45
    // Output window level statistics
    [(COUNT(sd) > 0) ? PROPERTY(OdEncPo_score,DFM_SCORE)/COUNT(sd) : 0]
    // Sum of values normalized by S/D count
    RDB ONLY DV COORD NULL
    ANNOTATE [DFM_METRIC = "DFM_SCORE"]
    ANNOTATE [DFM_RULE = "Active_Encl_Gate"]
    ANNOTATE [DFM_LEVEL = "window"]
}

```

```
ANNOTATE [DFM_TYPE = "cfa"]
ANNOTATE [DFM_BIN = ""]
ANNOTATE [DFM_GROUP = "Perf"]
ANNOTATE [DFM_PRIORITY = "2-MC"]
ANNOTATE [DFM_DESCRIPTION = "Increase the spacing from gate poly to
inside corner of active to improve this score"]

}
```


Appendix B

Foundry Certified Flows

Certified Calibre DFM functionality for each of the various foundry flows is described.

It also describes how to use this functionality with a foundry-supplied design kit.

Using Calibre With a UMC Rule File	424
Running Critical Area Analysis With a UMC Rule File	424
Running Calibre YieldEnhancer With a UMC Rule File	426
Using Calibre With a Common Platform Alliance Rule File.....	427
Running Critical Area Analysis With a Common Platform Alliance Rule File.....	427
Running Critical Feature Analysis With a Common Platform Alliance Rule File.....	428
Using Calibre With an ST Design Kit	429
Running Critical Feature Analysis With an ST Rule File	429
Running Calibre YieldEnhancer With an ST Rule File	429

Using Calibre With a UMC Rule File

UMC has certified Calibre tools for performing critical area analysis and yield enhancement.

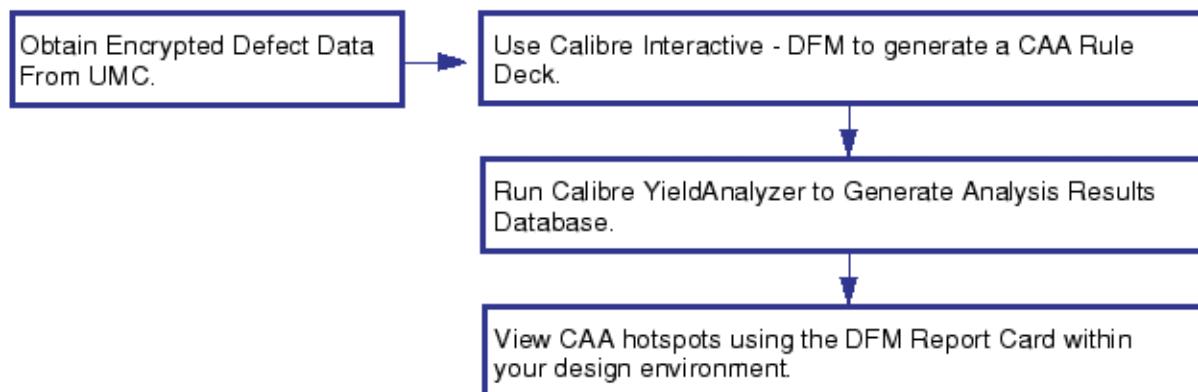
Running Critical Area Analysis With a UMC Rule File	424
Running Calibre YieldEnhancer With a UMC Rule File	426

Running Critical Area Analysis With a UMC Rule File

The Calibre YieldAnalyzer Critical Area Analysis (CAA) functionality is fully certified in the UMC flow.

- Users must generate the CAA rule deck through the YieldAnalyzer Critical Area Analysis user interface in Calibre Interactive DFM.
- The defect density data required for generating the CAA rule deck is provided by UMC as an encrypted file. The resulting rule deck is also encrypted.

Figure B-1. Calibre CAA with UMC Defect Data



Prerequisites

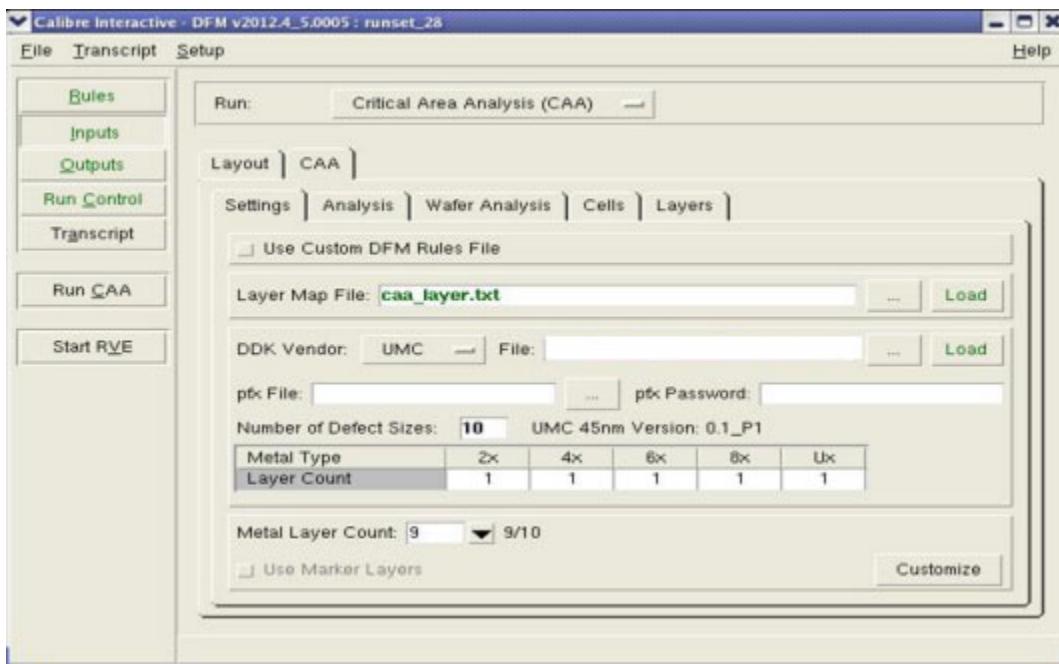
- The Calibre software tree is installed.
- The CALIBRE_HOME or MGC_HOME environment variable is set to point to the Calibre version software tree.
- You have obtained the UMC DDK.

Procedure

1. Create a CAA runset.

Try It! 	<p>Calibre Critical Area Analysis Tutorial and Example Kit</p> <p>Includes instructions on creating a runset along with documentation and data for performing Critical Area Analysis (CAA) with Calibre YieldAnalyzer. The procedures step you through running CAA on a chip, CAA with marker layers, CAA on a cell library, and CAA with No Defect Density (NDD).</p> <p>Go to this page on Support Center to download the eKit (Documentation tab, Document Types=Getting Started Guide). The link goes to the latest release.</p>
---	---

2. To load the layout and the defect density file, perform the following steps:



- In the Rules pane, browse to or type the path to the rule file (this is optional).
- In the Inputs pane, choose the **Layout** tab. Browse to or type the path to the layout file, and enter the top cell for the design.
- In the Outputs pane, enter the name of the desired output DFM database in the DFM Database text box.
- In the Inputs pane, choose the **CAA** tab. Under DDK Vendor, select UMC.
- In the File field, browse to or type the path to the defect data file you downloaded from UMC. Enter the number of defect sizes, number of 2xMetal layers, 4xMetal layers, and so on.

- f. In the pfx File text box, browse to or type the path to the pfx file you obtained from UMC. Supply the associated password in the pfx Password text box. Click **Load**.
- g. Choose the number of layers to process, then select the desired options in the **Analysis** tab.

Running Calibre YieldEnhancer With a UMC Rule File

You can run Calibre YieldEnhancer with a UMC rule file.

Prerequisites

- You have contacted your UMC representative to obtain a Calibre YieldEnhancer (YE) rule file.

Procedure

1. Run calibre -dfm with that rule file to make yield improving modifications to your design.
2. Backannotate your changes as described elsewhere in this manual.



Note

Backannotation is not yet certified for the UMC flow.

Using Calibre With a Common Platform Alliance Rule File

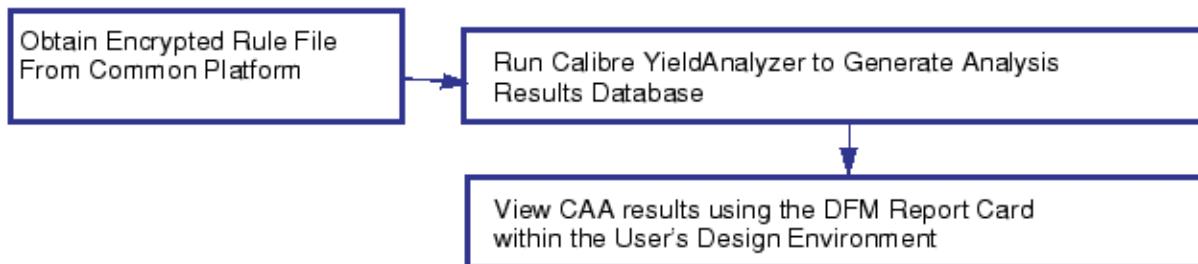
The Common Platform Alliance has developed rule files for performing critical area analysis and critical feature analysis. These rule files are available to their customers in encrypted form.

- Running Critical Area Analysis With a Common Platform Alliance Rule File** **427**
Running Critical Feature Analysis With a Common Platform Alliance Rule File **428**

Running Critical Area Analysis With a Common Platform Alliance Rule File

You can run CAA with a Common Platform Alliance rule file.

Figure B-2. Calibre CAA with a Common Platform Alliance Rule File



Prerequisites

- You have contacted your Common Platform Alliance representative to obtain an encrypted CAA rule file.

Procedure

1. Run calibre -drc with that rule file to identify the critical areas on your design.
2. Review the CAA results.

Try It! 	Calibre Critical Area Analysis Tutorial and Example Kit Includes documentation and data for performing Critical Area Analysis (CAA) with Calibre YieldAnalyzer. The procedures step you through running CAA on a chip, CAA with marker layers, CAA on a cell library, and CAA with No Defect Density (NDD). Go to this page on Support Center to download the eKit (Documentation tab, Document Types=Getting Started Guide). The link goes to the latest release.
--------------------	---

Running Critical Feature Analysis With a Common Platform Alliance Rule File

You can run CFA with a Common Platform Alliance rule file.

Prerequisites

- You have contacted your Common Platform Alliance representative to obtain an encrypted CFA rule file.

Procedure

1. Run `calibre -dfm` with that rule file to identify the critical areas on your design.
2. Review the CFA results.

Try It!	Calibre Critical Feature Analysis Tutorial and Example Kit Includes documentation and data for performing Critical Feature Analysis (CFA) with the Calibre YieldAnalyzer and Calibre YieldEnhancer tools. This tutorial walks you through building an analysis database, assessing design quality, locating and prioritizing fixes for the worst problems, and locating and prioritizing fixes for areas most likely to fail. Go to this page on Support Center to download the eKit (Documentation tab, Document Types=Getting Started Guide). The link goes to the latest release.
---------	---

Note

-  The CFA rule deck provided through the design kit from the Common Platform Alliance is written using only Calibre YieldAnalyzer, and therefore does not provide users with improbability data. Contact your Siemens EDA representative if you would like more information.
-

Using Calibre With an ST Design Kit

ST has developed rule files for performing critical feature analysis and yield enhancement. These rule files are available to their customers.

Running Critical Feature Analysis With an ST Rule File	429
Running Calibre YieldEnhancer With an ST Rule File.....	429

Running Critical Feature Analysis With an ST Rule File

You can run CFA with an ST rule file.

Prerequisites

- You have contacted your ST representative to obtain a CFA rule file.

Procedure

1. Run calibre -dfm with that rule file to identify the critical areas on your design.
2. Review the CFA results.

Try It! 	Calibre Critical Feature Analysis Tutorial and Example Kit Includes documentation and data for performing Critical Feature Analysis (CFA) with the Calibre YieldAnalyzer and Calibre YieldEnhancer tools. This tutorial walks you through building an analysis database, assessing design quality, locating and prioritizing fixes for the worst problems, and locating and prioritizing fixes for areas most likely to fail. Go to this page on Support Center to download the eKit (Documentation tab, Document Types=Getting Started Guide). The link goes to the latest release.
---	---

Running Calibre YieldEnhancer With an ST Rule File

You can run Calibre YieldEnhancer with an ST rule file.

Prerequisites

- You have contacted your ST representative to obtain a Calibre YieldEnhancer (YE) rule file.

Procedure

1. Run calibre -dfm with that rule file to make yield improving modifications to your design.

2. Backannotate your changes as described elsewhere in this manual.

Index

— Symbols —

- , 258
!, 258
[], 26
{}, 26
+, 258
|, 26
~, 258

— A —

ALLDEPTH, 264
ASCII databases, 268
Assessing yield, 270

— B —

Binning, 240
 random particles, 240
Bold words, 26
BOSE-EINSTEIN, 271
BY CELL, 263

— C —

CAA with No Defect Density, 117
Calibre DRC hierarchical engine
 results, 241
Calibre results, 241
Calibre transcript, 243
Calibre YieldAnalyzer
 benefits, 17
 overview, 17, 20
Calibre YieldEnhancer
 benefits, 17
 overview, 17, 22
Capture windows, 244
Cell method, 263
Chunking for statistical evaluation, 259
Clusters
 error, 251
 polygon, 245
Concurrency

defined, 245
Conditional expressions, 254
Courier font, 26
Critical area
 and quality assessment metrics, 265
 definition, 247
 for opens, 249
 for shorts, 247

— D —

Databases
 ASCII, 268
 GDS, 267
 OASIS, 268
 results, 25
 used by Calibre DRC hierarchical engine, 25
Defect density
 definition, 250
Design flow
 LFD, 20
DFM properties, 256
DFM RDB (results database), 269
DFM Spec Via Shift, 275
DFM Via Shift, 280
DFM Via Shift commands, 274
Double pipes, 26
DRC CHECK MAP, 42
DRC check results, 54
DRC results databases, 267
DRC results summary, 241

— E —

Edge clusters, 251
Error layers, 251
Event log, 243
EXPONENTIAL (yield model), 271
Expression chains, 254

— F —

Flows

LFD in design, 20

— G —

GDS databases, 267

GDS Files

as input to Calibre, 25

— H —

Heavy font, 26

— I —

INSIDE OF, 262

Invocation, 30

Italic font, 26

— L —

Lambda, 271

LFD

in the design flow, 20

Limiting the scope of an operation, 262

— M —

Manufacturability

predicting, 265

Metrics

for assessing yield, 270

to assess design quality, 265

Minimum keyword, 26

Models

yield, 271

MURPHY (yield model), 271

— N —

NEGATIVE-BINOMIAL, 271

NOHIER, 264

Non-parallel wires

shorts between, 248

NOPSEUDO, 264

Not, 258

— O —

OASIS databases, 268

Opens

at vias, 248

critical area for, 249

illustrated, 247, 249

Operations

concurrent, 245

Operators

binary, 257

unary, 258

Organizing data, 259

— P —

Parentheses, 26

Partitioning, 259

by cell, 263

by polygon cluster, 245

by window, 260

Pipes, 26

POISSON, 271

Polygon clusters, 245

Predicting manufacturability, 265

Product overview, 17, 18, 20

Properties, 256

used to save DRC check results, 53

— Q —

Quality assessment metrics, 265

Quotation marks, 26

— R —

Random particles

binning of, 240

probability of, 250

RDB

cell method, 263

window method, 259

Recommended rule violations

and quality assessment metrics, 265

Redundancy

and quality assessment metrics, 265

Results database

definition, 267

methods for generating, 267

types of, 267

Runsets, 269

— S —

SAME CELL

with DFM RDB, 264

Save

nmDRC check results, 53

Shorts

critical area for, 247

Slanted words, [26](#)
Square parentheses, [26](#)
Statistics, [240](#)
STEP keyword, [261](#)
SVRF statements
 DRC CHECK MAP, [42](#)

— **T** —

Transcript, Calibre, [243](#)

— **U** —

Underlined words, [26](#)
USER (yield model), [271](#)

— **W** —

Warning messages, [243](#)
WINDOW boundary, [262](#)
Window method, [259](#)
WINDOW size, [261](#)

— **Y** —

Yield assessment metrics, [270](#)
Yield models, [271](#)
YieldAnalyzer described, [17](#), [20](#)
YieldEnhancer described, [17](#), [20](#)

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.

