SIEMENS EDA

Calibre® Solutions for Physical Verification

Software Version 2021.2
Document Revision 14

SIEMENS

Unpublished work. © 2021 Siemens

This material contains trade secrets or otherwise confidential information owned by Siemens Industry Software, Inc., its subsidiaries or its affiliates (collectively, "Siemens"), or its licensors. Access to and use of this information is strictly limited as set forth in Customer's applicable agreement with Siemens. This material may not be copied, distributed, or otherwise disclosed outside of Customer's facilities without the express written permission of Siemens, and may not be used in any way not expressly authorized by Siemens.

This document is for information and instruction purposes. Siemens reserves the right to make changes in specifications and other information contained in this publication without prior notice, and the reader should, in all cases, consult Siemens to determine whether any changes have been made. Siemens disclaims all warranties with respect to this document including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement of intellectual property.

The terms and conditions governing the sale and licensing of Siemens products are set forth in written agreements between Siemens and its customers. Siemens' **End User License Agreement** may be viewed at: www.plm.automation.siemens.com/global/en/legal/online-terms/index.html.

No representation or other affirmation of fact contained in this publication shall be deemed to be a warranty or give rise to any liability of Siemens whatsoever.

TRADEMARKS: The trademarks, logos, and service marks ("Marks") used herein are the property of Siemens or other parties. No one is permitted to use these Marks without the prior written consent of Siemens or the owner of the Marks, as applicable. The use herein of third party Marks is not an attempt to indicate Siemens as a source of a product, but is intended to indicate a product from, or associated with, a particular third party. A list of Siemens' trademarks may be viewed at: www.plm.automation.siemens.com/global/en/legal/trademarks.html. The registered trademark Linux® is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a world-wide basis.

Support Center: support.sw.siemens.com
Send Feedback on Documentation: support.sw.siemens.com/doc_feedback_form

Revision History

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

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

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

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table of Contents

Revision History

Chapter 1
Introduction 13

Licensing 13
Syntax Conventions 13

Chapter 2
Advanced DRC Solutions 15

Contact and Via Checks 17
Counting Contacts in Source-Drain and Gate Regions 17
Checking Contact Balance 19
Verifying Contact Run Length for Current Crowding 21
Verifying Contact Placement for Current Crowding 22
Checking Maximum Via Coverage for Metal Overlap Regions 25
Device and Routing Checks 29
Checking Device Orientation 29
Checking Gate Width Dependent Via Count 32
Checking Gate Width Dependent Via Count With DFM Property CONNECTED 32
Checking Gate Width Dependent Via Count With DFM Property NODAL and Expand Edge 35
Checking Gate Width Dependent Via Count With DFM Property NODAL and DFM Stamp 37
Writing a Wide Metal Spacing Check 40
Writing Wide Metal Spacing Checks With Traditional DRC 40
Writing Wide Metal Spacing Checks Using a Continuous Function 42
Writing Wide Metal Spacing Checks Using DFM Function 45
Finding the Length of a Net Path Connected to Gate 48
Finding the Length of a Net Path Connected to Gate: Effective Length Method 48
Finding the Length of a Net Path Connected to Gate: Centerline Method 51
Comparison of Centerline and Effective Length Methods for Finding Path Length 55
Checking Unsilicided Resistors 57
Grid and Pitch Checks 59
Checking Polygon Centerline Alignment 59
Checking Equal Gate Pitch per Active Area 62
Checking Gate Relative Pitch per Active Area 64
Checking Metal Routing Relative Pitch 66
Checking Contact Relative Pitch 67
Miscellaneous Advanced DRC Checks 70
Checking Dangling Wires 71
Finding Circular Polygons 76
Finding Length and Width Edges of Rectangles 78

Finding Pad Edges Closest to the Chip Edge 81
Finding Row and Column Counts in an Array 83
Finding Polygon Extents for Special Configurations 85
Finding Unique Violations 87
Resizing Polygons by a Percentage of the Area 88

Chapter 3
Chip-Level Verification 93

Running Calibre nmDRC Recon 94
Selecting Checks in Calibre nmDRC Recon 95
Running Calibre DRC Analyze 98
Calibre RVE Feature Overview 99
Reviewing DRC Analyze Results 100
Running Calibre nmLVS Recon SI 107
Running Calibre nmLVS Recon Softchk. 109

Chapter 4
Density Best Practices. 111

Analyzing Density Results with Calibre RVE. 111
Density Operation Categories 117
Finding Density Outside of a Cell or Layer 117
Finding Density Inside of a Layer 119
Finding Density Inside of a Layer and Outside of a Cell 123
Reducing RDB Data Size with Density COMBINE 126
Using INSIDE OF LAYER Options. 129

Chapter 5
Empty Layers 135

Specification of an Empty Layer 135
Derivation of Empty Layers 135
Empty Layer Checks 136

Chapter 6
Layout Magnification 139

Considerations When Magnifying Data 139
Layout Database Versus Calibre Database 140
Key Specification Statements for Magnification. 141
Magnification and Precision Management 144
Scaling and Precision Differences 144
Effects on Coordinates 145
Magnification Values Less Than 1 147
Increasing Measurement Precision 147
Reducing Layout Magnification 149
Degenerate Boundary and Polygon Management 150

Chapter 7
Layout Rotation-Sensitive Commands 153
SVRF Statements Sensitive to Rotation 153
Litho Commands Sensitive to Rotation 154

Chapter 8
DRC Regression Testing 155
Generating XOR Regression Files with Calibre DESIGNrev 156

Chapter 9
Pattern Matching Applications 165
Capturing a Pattern From the Layout 165
Adding Layout Shapes Using Pattern Marker Output 168
Adding Constraints to Pattern Edges and the Output Marker 172
Converting DBCLASSIFY to DFM Pattern Classify for Result Classification 176

Chapter 10
Calibre RVE HTML Reporting. 181
DRC HTML Report for Waiver Flow 181
DRC HTML Report for Calibre PERC LDL CD 189
DRC HTML Report for Calibre LFD 191

Chapter 11
Device Counts 195
Flat Device Counts From Rule Checks 195
Device Counts in Circuit Extraction and Calibre PERC 196

Chapter 12
Electrical Rule Checks 199
MOSFET Bulk Pin Connections 200
MOSFET Power Domain Connections 201
Back-to-Back Diode Ground Domain Separation 202
ESD Device Protection of I/O Pads 203
ESD Resistor Protection of Gates 204
Pin Voltage Checks 205
Current Density and Resistance Checks 206

Chapter 13
LVS Best Practices 207
Recommended LVS Rule File Statements. 207
LVS Rule File Elements to Avoid 210
Hcell Recommendations for Digital Place and Route Flows 211
Efficient User-Defined Device Property Programs 213
Extracting Device Properties Only When Necessary 213
Using Local Property Measurements 214
LVS Box Statement Usage 215
Performing Only Short Isolation 218

Basic LVS Rule File Template 220

Chapter 14
Rule File Management: Environment Variables and Rule File Variables 223

Set Environment Variables and Run Calibre From Custom Menu Item in Design Tool . . . 223
Set Environment Variables with Calibre Interactive 225
Insert Rule File Variable and #DEFINE Statements with the Customization GUI. 229
Insert a Boolean #DEFINE Based on an Environment Variable 230
Insert a #DEFINE with a Value Based on an Environment Variable. 232
Setting a Rule File Variable with the Customization GUI 234

Chapter 15
Pre- and Post-Processing for Calibre Runs 237

Creating a One-Click Custom Menu Command With SKILL Pre-Processing 238
Processing a Netlist Before the Calibre Run 240

Index

Third-Party Information

List of Figures

Figure 2-1. Number of Contacts in Source-Drain Regions 18
Figure 2-2. Number of Contacts Associated with Gates 19
Figure 2-3. Contact Balance Concept 19
Figure 2-4. Contact Run Length Current Crowding 21
Figure 2-5. Contact Placement Current Crowding 23
Figure 2-6. Vias in a Metal Overlap Region 25
Figure 2-7. Sample Layout With Device Markers. 30
Figure 2-8. Gate Width Dependent Via Count 32
Figure 2-9. Gate Width Dependent Via Count Output 35
Figure 2-10. Erosion Effects 40
Figure 2-11. Case 1 41
Figure 2-12. Case 2 42
Figure 2-13. Case 3 42
Figure 2-14. Traditional DRC Versus eqDRC. 43
Figure 2-15. Wide Metal Spacing Output in Calibre DESIGNrev 45
Figure 2-16. Wide Metal Spacing Output 47
Figure 2-17. Net Path Connected to Gate 49
Figure 2-18. Net Path Connected to Gate 52
Figure 2-19. Comparison of Path Length Computation Methods 56
Figure 2-20. Unsilicided Resistor 57
Figure 2-21. Check Centerline Alignment. 60
Figure 2-22. Gate Pitch 62
Figure 2-23. Centerline Gate Pitch 64
Figure 2-24. Relative Pitch Checking for Metal Routing 66
Figure 2-25. Contact Relative Pitch. 68
Figure 2-26. Dangling Wires 71
Figure 2-27. DFM Circle Analyze Results 77
Figure 2-28. Inside Cell CIRCLE Results 78
Figure 2-29. Length and Width Edges of Rectangles 79
Figure 2-30. Pad Edges Closest to the Chip Edge 81
Figure 2-31. Finding Row and Column Counts. 85
Figure 2-32. Overlapping Polygon Extents 85
Figure 2-33. Example of Unique Spacing Violations Between Polygons. 87
Figure 4-1. Density Outside of a Cell 118
Figure 4-2. Density Inside of Marker Layer 120
Figure 4-3. Density Inside of Marker Layer (Improved). 121
Figure 4-4. Density Using Minimum Marker Layer Overlap 122
Figure 4-5. Windows Inside Marker Layer 123
Figure 4-6. Density Inside of Boundary Layer and Outside a Cell 124
Figure 4-7. Windows Inside Boundary Layer 125

Figure 4-8. Combined Density RDB Results. 127
Figure 4-9. Density COMBINE Results Highlighting. 127
Figure 4-10. DAVG, DMIN, and DMAX Row Highlights 128
Figure 4-11. Results Using COMBINE <= 20 128
Figure 4-12. Density INSIDE OF LAYER marker 130
Figure 4-13. Density INSIDE OF LAYER marker CENTERED 131
Figure 4-14. Density INSIDE OF LAYER marker BY POLYGON. 132
Figure 4-15. Density INSIDE OF LAYER marker BY RECTANGLE 133
Figure 6-1. Discrete Database Coordinates 139
Figure 6-2. Degenerate Polygon Due to Precision Change 151
Figure 8-1. DRC Regression Flow 156
Figure 9-1. Example Layout for Pattern Matching 165
Figure 9-2. Layout Correction with Pattern Marker Output 168
Figure 10-1. Calibre RVE View of Waiver Flow Results 182
Figure 10-2. DRC HTML Report for Waiver Flow. 184
Figure 10-3. DRC HTML Report for PERC LDL CD 190
Figure 10-4. DRC HTML Report for Calibre LFD 192
Figure 12-1. MOSFET Bulk Pin Connections. 200
Figure 12-2. MOSFET Connections to Differing Power Domains 201
Figure 12-3. Back-to-Back Diode Separation 202
Figure 12-4. I/O Pad With ESD Device Protection 203
Figure 12-5. ESD Resistor Protection of Gates 204
Figure 12-6. Pin Voltage Checks 205
Figure 12-7. Current Density and Resistance 206
Figure 14-1. Custom Menu Item in Design Tool. 224
Figure 14-2. Customization GUI in Calibre Interactive 229
Figure 14-3. Customization GUI Results for Boolean #DEFINE 231
Figure 14-4. Customization GUI Results for Boolean Define in TVF Rule File 232

List of Tables

Table 1-1. Syntax Conventions 13
Table 2-1. Contact and Via Check Solutions 17
Table 2-2. Device-Level and Routing Check Solutions 29
Table 2-3. Comparison of Path Length Computation Methods 56
Table 2-4. Grid and Pitch Check Solutions 59
Table 2-5. Miscellaneous DRC Check Solutions 70
Table 3-1. Keywords for Rule Check Groups in Calibre nmDRC Recon 96
Table 3-2. Calibre RVE for DFM Results Tabs 99
Table 4-1. Density Operation Solutions 117
Table 5-1. Global Empty Layer Checks 137
Table 6-1. Specification Statements Affecting Magnification 141
Table 7-1. Rotation-Sensitive SVRF Statements 153
Table 7-2. Rotation-Sensitive SVRF Commands Due to Grid Snapping 154
Table 7-3. Rotation-Sensitive Litho Commands 154
Table 13-1. Recommended LVS Rule File Statements 208
Table 13-2. LVS Rule File Elements to Avoid 210
Table 13-3. Differences Between LVS Box Flows 215
Table 15-1. Pre- and Post-Processing Methods with Calibre Verification 237

Chapter 1
Introduction

This manual contains Calibre solutions to common IC physical verification problems from a broad spectrum of topics.

IC design and CAD engineers supporting physical verification flows will find information in this manual useful.

Licensing 13
Syntax Conventions 13

Licensing

The licenses required to perform the procedures discussed in this manual include Calibre® Interactive™, Calibre® nmDRC™/Calibre® nmDRC-H™, Calibre® nmLVS™/ Calibre® nmLVS-H™, Calibre® PERC™, Calibre® RVE™, and Calibre® YieldServer.

Complete licensing information is provided under "Licensing: Physical Verification Products" in the *Calibre Administrator's Guide*. For the Bridge Fault ATPG flow, consult the Tessent documentation for licensing information pertaining to that tool set.

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.
Italic	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.
Underline	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.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 1-1. Syntax Conventions (cont.)

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

Example:

DEVice {*element_name* [‘(’*model_name*‘)’]}

device_layer {***pin_layer*** [‘(’*pin_name*‘)’] …}

[‘<’*auxiliary_layer*‘>’ …]

[‘(’*swap_list*‘)’ …]

[BY NET | BY SHAPE]

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Chapter 2
Advanced DRC Solutions

Advanced DRC solutions use DFM properties to classify shapes in the design. Many of the solutions use equation-based DRC (eqDRC), which is a set of SVRF statements that extend the capabilities of Calibre nmDRC. Using eqDRC, you can analyze complex interactions that are difficult or impossible to verify using traditional DRC methods. The eqDRC toolset includes the ability to manipulate DFM properties, and to write complex equations using DFM Expressions.

Try It!	**Calibre Advanced DRC (eqDRC) Tutorial and Example Kit**
	This example kit is designed to assist users of Calibre eqDRC and those writing advanced DRC rule checks. It contains the data and rule files necessary to run all the examples in this chapter. Go to this page on Support Center to download the complete eKit. The link goes to the latest release (Documentation tab, Document Types=Getting Started Guide, Categories=IC Verification & Signoff).

Contact and Via Checks **17**
Counting Contacts in Source-Drain and Gate Regions 17
Checking Contact Balance 19
Verifying Contact Run Length for Current Crowding 21
Verifying Contact Placement for Current Crowding 22
Checking Maximum Via Coverage for Metal Overlap Regions 25

Device and Routing Checks **29**
Checking Device Orientation 29
Checking Gate Width Dependent Via Count 32
Writing a Wide Metal Spacing Check 40
Finding the Length of a Net Path Connected to Gate 48
Checking Unsilicided Resistors 57

Grid and Pitch Checks **59**
Checking Polygon Centerline Alignment 59
Checking Equal Gate Pitch per Active Area 62
Checking Gate Relative Pitch per Active Area 64
Checking Metal Routing Relative Pitch 66
Checking Contact Relative Pitch 67

Miscellaneous Advanced DRC Checks **70**
Checking Dangling Wires 71
Finding Circular Polygons 76
Finding Length and Width Edges of Rectangles 78

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Finding Pad Edges Closest to the Chip Edge 81
Finding Row and Column Counts in an Array 83
Finding Polygon Extents for Special Configurations 85
Finding Unique Violations 87
Resizing Polygons by a Percentage of the Area 88

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Contact and Via Checks

This section presents various contact and via checks.

Table 2-1. Contact and Via Check Solutions

Name	Description	SVRF Elements
Counting Contacts in Source-Drain and Gate Regions	Counts the number of contacts in the source-drain region.	AND, DFM Property, Interact, NOT
Checking Contact Balance	Checks that source-drain regions on both sides of a gate have an equal number of contacts, but allows a difference of one contact if both source-drain regions each have at least two contacts.	DFM Function, DFM Property, DFM RDB
Verifying Contact Run Length for Current Crowding	Quantifies current crowding through a gate based on the run length of contacts on each side of the gate and identifies gates that have the worst current crowding effects.	Coincident Edge, DFM Property, DFM RDB, External, Length
Verifying Contact Placement for Current Crowding	Checks for current crowding effects based on the number of contacts and their positions in the source and drain regions.	AND, Area, Interact, DFM Copy, DFM Property, DFM RDB, External
Checking Maximum Via Coverage for Metal Overlap Regions	Checks that metal overlap regions contain the maximum possible number of vias.	DFM Property, DFM RDB, Variable

Counting Contacts in Source-Drain and Gate Regions

You can use a multilayer DFM Property operation to return the number of contacts in each source-drain region and each gate region.

The process first counts the number of contacts in each source-drain region and saves this as the property sd_con attached to the source-drain shapes on the layer sd_count.

The source-drain regions abut the gate shapes, so a multilayer DFM Property operation with the ABUT ALSO keyword is used to sum the contacts in each source-drain region abutting a gate. The MULTI keyword is needed because a source-drain region can be associated with more than one gate geometry. If you do not specify MULTI, each source-drain region is associated with a single gate, which Calibre determines arbitrarily, and a warning is issued in the transcript. The

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

PROPERTY() function is used to read the sd_con property value attached to each source-drain region.

Try It!	**Calibre Advanced DRC (eqDRC) Tutorial and Example Kit**
	Go to this page on Support Center to download the complete eKit.
	This example is in the contact_via_checks/sd_count example directory.

Procedure

1. Derive the layers for recognizing gate and source-drain regions:

```
LAYER CON               // contact layer
gate = POLY AND AA
imp = AA NOT POLY
sd = imp INTERACT POLY
```

2. Count the contacts in each source-drain region using a DFM Property operation:

```
sd_count = DFM PROPERTY sd CON INTERSECTING
               [sd_con = COUNT(CON)]
```

3. Sum the contacts in the source-drain regions associated with each gate.

```
gate_count = DFM PROPERTY gate sd_count OVERLAP ABUT ALSO MULTI
                 [gate_con = PROPERTY(sd_count,sd_con)]
```

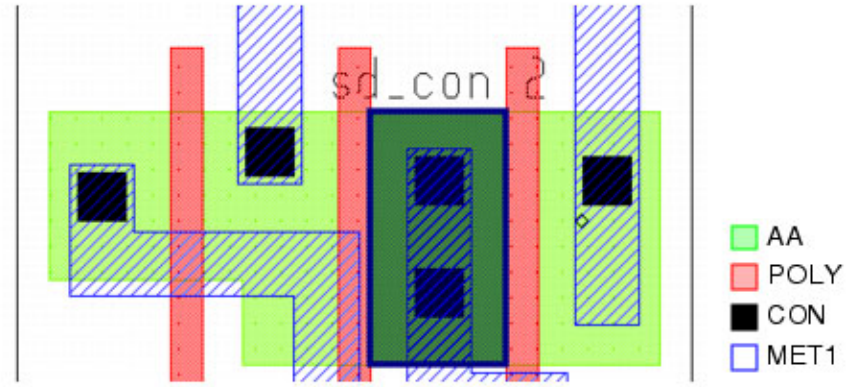
4. Output the results of the DFM Property operation using DFM RDB.

```
contact_counts {
    DFM RDB sd_count eqdrc.rdb CHECKNAME "%_l_"
        ALL CELLS CELL SPACE
    DFM RDB gate_count eqdrc.rdb CHECKNAME "%_l_"
        ALL CELLS CELL SPACE
}
```

Results

The following figure shows an sd_count result in Calibre DESIGNrev where the count is 2.

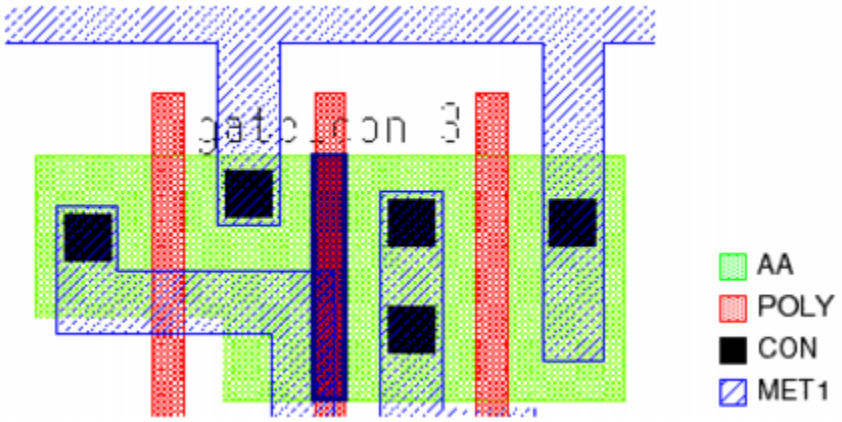
Figure 2-1. Number of Contacts in Source-Drain Regions



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following figure shows a gate_count result in Calibre DESIGNrev where the number of contacts associated with the highlighted gate is 3.

Figure 2-2. Number of Contacts Associated with Gates



Related Topics

Introduction to DFM Property [Calibre Verification User's Manual]

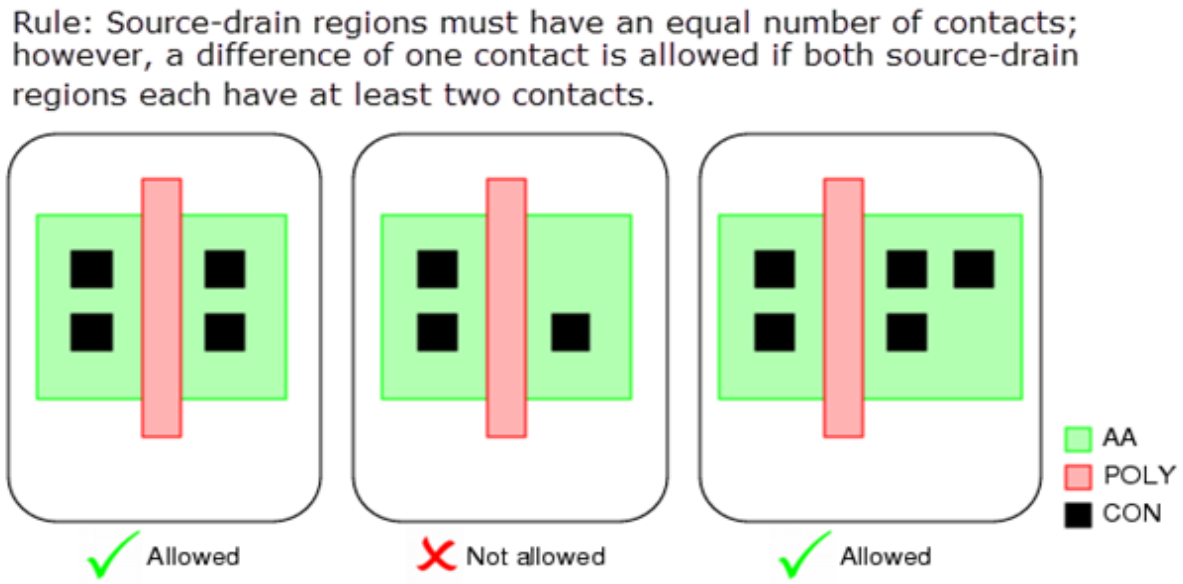
Measurement Functions [Standard Verification Rule Format (SVRF) Manual]

Introduction to DFM RDB [Calibre Verification User's Manual]

Checking Contact Balance

You can check that the contacts in the source-drain regions on both sides of a gate are in balance.

Figure 2-3. Contact Balance Concept



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The logic for this rule is implemented using conditional expressions and a function that is defined in a DFM Function statement.

Try It!	Calibre Advanced DRC (eqDRC) Tutorial and Example Kit
	Go to this page on Support Center to download the complete eKit. This example is in the contact_via_checks/contact_balance example directory.

Procedure

1. Define a DFM Function named “unbalanced” that accepts two numeric values and evaluates a condition. The function returns a 0 if the condition passes and a 1 if it fails. To do this, use a nested conditional expression:

```
DFM FUNCTION [ unbalanced(NUMBER nmax, NUMBER nmin)
   (nmin >= 2) ? (( nmax - nmin <= 1 ) ? 0 : 1 ) :
   (( nmax == nmin ) ? 0 : 1 ) ]
```

   The two evaluations in the conditional expression read as follows:

   - If the minimum number of contacts for either source-drain region is at least 2, the difference in the number of contacts must be less than or equal to 1. If this statement evaluates true, return 0. If this statement evaluates false, return 1.
   - Else, if the minimum number of contacts for either source-drain region is less than 2, the number of contacts for both source-drain regions must be equal. If this statement evaluates true, return 0. If this statement evaluates false, return 1.

2. Derive a layer that counts the number of contacts in each source-drain region using a DFM Property operation:

```
sdcont = DFM PROPERTY sd CON [N = COUNT(CON)]
```

3. Derive a layer that identifies gates that do not pass the condition:

```
contact_balance = DFM PROPERTY gate sdcont OVERLAP ABUT ALSO MULTI
                  [ - = unbalanced(MAX(PROPERTY(sdcont, N)),
                        MIN(PROPERTY(sdcont, N)))] > 0
```

   This is how the rule check works:

   - A non-persistent DFM property is used to identify failing gates without attaching property values to them.
   - The source-drain regions are clustered with each gate that they overlap or abut. The MIN and MAX measurement functions and the PROPERTY access functions are used to pass the minimum and maximum number of contacts (the N property) to the DFM Function named “unbalanced” from Step 1.
   - A constraint value of 0 causes output when the unbalanced function returns 1.

4. Output the results using the DFM RDB operation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
eqdrc_out {
   DFM RDB contact_balance eqdrc.rdb CHECKNAME "%_l_"
       CELL SPACE ALL CELLS
}
```

Related Topics

Introduction to DFM Function [Calibre Verification User's Manual]

Introduction to DFM RDB [Calibre Verification User's Manual]

Conditional Expressions [Standard Verification Rule Format (SVRF) Manual]

Non-Persistent DFM Properties [Standard Verification Rule Format (SVRF) Manual]

Introduction to DFM Property [Calibre Verification User's Manual]

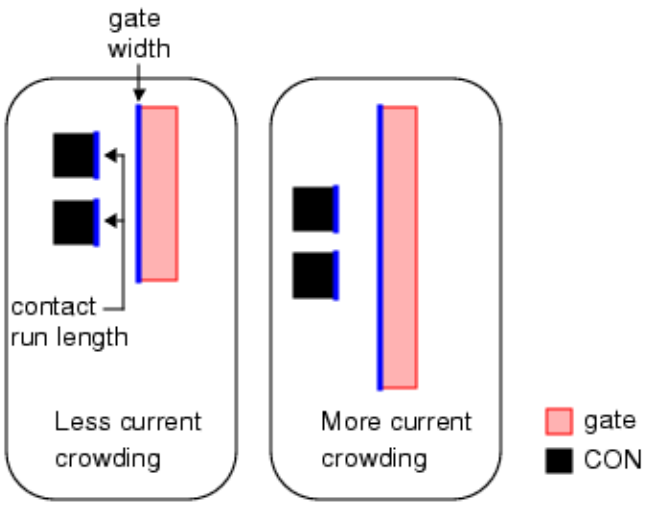
Measurement Functions [Standard Verification Rule Format (SVRF) Manual]

Verifying Contact Run Length for Current Crowding

You can quantify current crowding through a gate based on the run length of contacts on each side of the gate and identify gates that have the worst current crowding effects.

This rule uses the ratio of gate width to total contact run length. As this ratio increases, the current density around the contacts is likely to increase, causing an undesirable increase in the series resistance of the device. See Figure 2-4.

Figure 2-4. Contact Run Length Current Crowding



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Try It!	**Calibre Advanced DRC (eqDRC) Tutorial and Example Kit**
	Go to this page on Support Center to download the complete eKit.
	This example is in the contact_via_checks/ current_crowding_con_run_length example directory.

Procedure

1. Derive the contact to gate edge pairs:

```
con_gate_spc = EXT CON gate < 0.15 OPPOSITE
```

2. Derive the gate edges:

```
gate_edges = COIN EDGE gate sd
```

3. Use the LENGTH function to measure the length of the gate edge, and the EC function to measure the common run length of the contact-to-gate edge pairs. Divide these quantities to obtain a ratio:

```
current_crowding = DFM PROPERTY gate_edges con_gate_spc
                     OVERLAP ABUT ALSO
                     [ratio = LENGTH(gate_edges)/EC(con_gate_spc)]
```

   A high ratio suggests increased current crowding.

4. Output the results using a DFM RDB operation.

```
current_crowding {
  DFM RDB current_crowding eqdrc.rdb CHECKNAME "%_l_"
        CELL SPACE ALL CELLS
}
```

Related Topics

Introduction to DFM Property [Calibre Verification User's Manual]

Measurement Functions [Standard Verification Rule Format (SVRF) Manual]

Introduction to DFM RDB [Calibre Verification User's Manual]

Math Operators [Standard Verification Rule Format (SVRF) Manual]

Verifying Contact Placement for Current Crowding

You can check for current crowding effects based on the number of contacts and their positions in the source and drain regions.

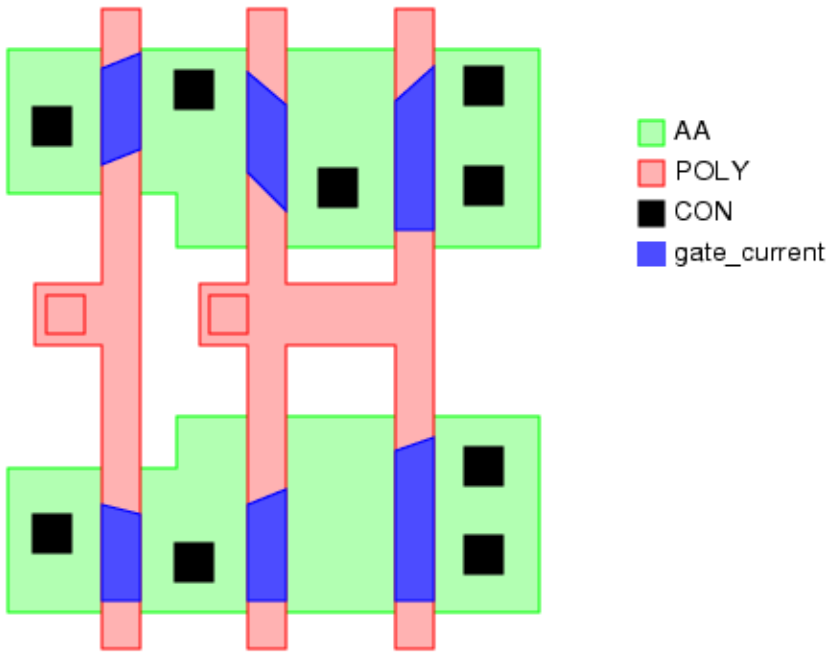
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

This rule check generates and processes non-rectangular (skew) polygons. In general, processing large numbers of skew polygons is not recommended, as it can degrade performance.

The rules set developed in this procedure mimics the actual flow of current between contacts across one or more gates by producing polygons on each gate. The current that flows through each gate is represented visually by the area of each of these polygons. See the following figure.

Figure 2-5. Contact Placement Current Crowding



Try It!	**Calibre Advanced DRC (eqDRC) Tutorial and Example Kit**
	Go to this page on Support Center to download the complete eKit.
	This example is in the contact_via_checks/ current_crowding_con_placement example directory.

Procedure

1. Derive the gate layer, source-drain (sd) layer, and the contacts in the source and drain regions.

```
gate = POLY AND AA
sd = (AA NOT POLY) INTERACT gate
con_and_sd = CON AND sd
```

2. Derive the source and drain regions that contain one or more contacts.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
sd_w_con = DFM PROPERTY sd con_and_sd
                    [- = COUNT(con_and_sd)] > 0
```

3. Derive an edge layer containing the contact edges that face each gate. Use a constraint based on the minimum gate spacing.

```
VARIABLE min_gate_spacing 0.11
a1 = EXT [con_and_sd] gate < 1.5*min_gate_spacing OPPOSITE
```

4. Derive an error layer containing pairs of contact edges that span across one or more gates (that is, one or more gates are between the edges in an edge pair). The constraint distance should be at least twice min_gate_spacing plus the gate width. Use a larger constraint distance to include results such as the lower right and center results in Figure 2-5.

```
a2 = EXT a1 < 8*min_gate_spacing
```

5. Only consider a2 edge pairs that cross only one gate and do not span more than one active area region.

```
a3 = DFM PROPERTY a2 sd_w_con gate AA REGION OVERLAP MULTI
       [- = COUNT(sd_w_con)] == 2 // pairs that span only two sd
                                  // regions with contacts, that is,
                                  // cross only one gate.
       [- = COUNT(gate)] > 0  // pairs that cross at least one gate
       [- = COUNT(AA)] == 1   // pairs that cross only one AA region
```

6. Derive the gate overlap regions.

```
a4 = DFM COPY a3 REGION
a5 = a4 AND gate
```

7. Expand gate regions to account for current spreading effects.

```
a6 = SIZE a5 BY 0.08 INSIDE OF gate
```

8. For each gate, use a DFM Property operation to calculate the percentage of the gate not covered by the overlap region.

```
gate_current = DFM PROPERTY a6 gate OVERLAP MULTI
    [pct_w_loss = ROUND(100*(AREA(gate)-AREA(a6)/AREA(gate)),0.1)]
```

9. Output the results of the DFM Property operation using DFM RDB.

```
gate_current {
   DFM RDB gate_current eqdrc.rdb CHECKNAME "%_l_"
       CELL SPACE ALL CELLS
}
```

Related Topics

Introduction to DFM Property [Calibre Verification User's Manual]

Measurement Functions [Standard Verification Rule Format (SVRF) Manual]

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Non-Persistent DFM Properties [Standard Verification Rule Format (SVRF) Manual]

Introduction to DFM RDB [Calibre Verification User's Manual]

Supported Math Functions in DFM Expressions [Standard Verification Rule Format (SVRF) Manual]

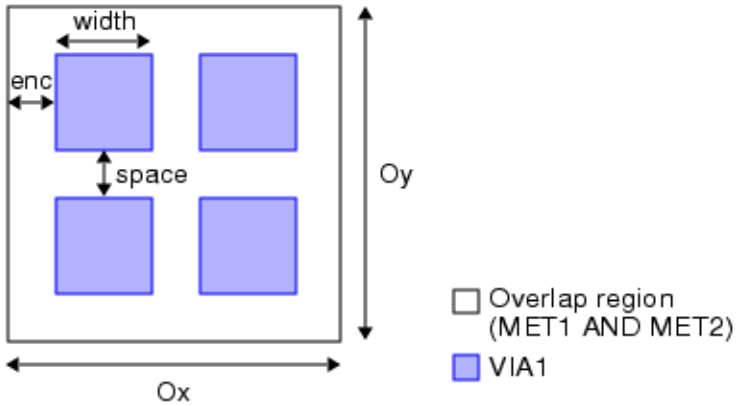
Math Operators [Standard Verification Rule Format (SVRF) Manual]

Checking Maximum Via Coverage for Metal Overlap Regions

You can identify metal overlap regions that contain fewer vias than the maximum number of vias possible based on the specified enclosure, width, and space constraints. You can also determine if the vias have the expected total area.

In the following figure, Ox and Oy define the extent of a region of overlap between two metal layers.

Figure 2-6. Vias in a Metal Overlap Region



For any rectangular overlap region, the maximum number of vias that can fit in the region is this:

$$maxvias = \frac{Oy - 2(enc) + space}{width + space} \times \frac{Ox - 2(enc) + space}{width + space}$$

The following procedure demonstrates how to write an eqDRC rule check for the number of vias and the via area. The via width, spacing, and enclosure rules are not considered, but these checks can be done with standard DRC rule checks.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Try It!	**Calibre Advanced DRC (eqDRC) Tutorial and Example Kit**
	Go to this page on Support Center to download the complete eKit.
	This example is in the contact_via_checks/max_via_coverage example directory.

Procedure

1. Define the variables and derive the extent of the overlap region:

```
VARIABLE _enc 0.05
VARIABLE _space 0.05
VARIABLE _width 0.15

met1_met2_overlap = EXTENTS ((MET1 AND MET2) ENCLOSE VIA1)
```

2. Use a multilayer DFM Property operation to calculate the maximum number of vias that can be placed in the overlap region. Also determine if the vias that are present have the expected area. The default INTERSECTING clustering is used so that only the portion of the vias within the metal overlap area are considered.

```
via_coverage = DFM PROPERTY met1_met2_overlap VIA1
    [ "+maxvias" = (FLOOR((((PERIMETERY(met1_met2_overlap)/2)-
                   (2*_enc)+_space)/(_width+_space))+0.0001)*
                   FLOOR((((PERIMETERX(met1_met2_overlap)/2)-
                   (2*_enc)+_space)/(_width+_space))+0.0001))]
    ["+vias_diff" = PROPERTY_REF("+maxvias") - COUNT(VIA1)]
    ["+ideal_area" = _width*_width*PROPERTY_REF("+maxvias")]
    [vias_missing = ( PROPERTY_REF("+vias_diff") >= 0 ) ?
                      PROPERTY_REF("+vias_diff") : 0 ]
    [vias_extra = ( PROPERTY_REF("+vias_diff") < 0 ) ?
                     ABS(PROPERTY_REF("+vias_diff")) : 0 ]

    // via_area_small is the fractional amount by which the
    // actual via area is smaller than the expected via area.
    // It is 0 if the actual via area is greater than or equal to
    // the expected via area.
    // An expression chain is used in case an +ideal_area of zero
    // causes divide by zero.
    // DRC_LT returns 1 if actual via area < ideal via area
    [via_area_small =
           (DRC_LT(AREA(VIA1), PROPERTY_REF("+ideal_area")) > 0) ?
               1 - (AREA(VIA1)/PROPERTY_REF("+ideal_area") ) : 0 ]
           [via_area_small = 0]
    // produce output if any value is greater than 0
    [- = PROPERTY_REF(vias_missing) + PROPERTY_REF(vias_extra)
         + PROPERTY_REF(via_area_small)] > 0
```

These properties are calculated:

- "+maxvias" — The maximum number of vias that can be placed in the metal overlap area. The PERIMETERX and PERIMETERY measurement functions return two parallel edges for each polygon, so it is necessary to divide these values by 2 to get the Ox and Oy values. The initial + in the property name marks it as a named non-

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

persistent property. The property can be used in property reference functions such as PROPERTY_REF(), but it is not attached to the shapes on the output layer.

- “+vias_diff” — The difference between the actual number of vias and “+maxvias”. This is a named non-persistent property.
- “+ideal_area” — The expected via area, assuming the maximum number of vias of the expected dimensions.
- vias_missing — The number of vias that are missing.
- vias_extra — The number of extra vias.
- via_area_small — This value is zero if the actual via area is equal to or greater than “+ideal_area”. Otherwise it is the fractional amount by which the actual via area is smaller than expected.

  The DRC_LT function is used to compare the actual via area to the expected area within a built-in tolerance.

  An expression chain is used to prevent evaluation failure due to divide by zero if “+ideal_area” is zero, which might happen for a very small enclosure area.

The final property calculation causes result output only if an error condition exists. A non-persistent property is used, so the actual value is not attached to the output.

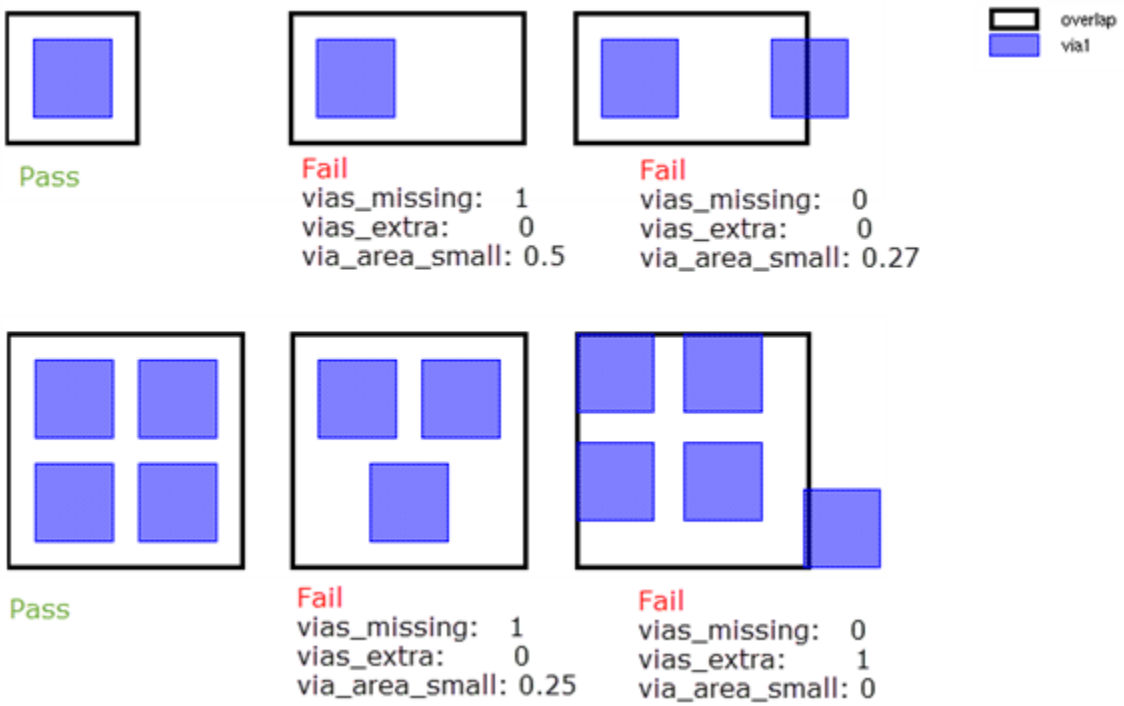
3. Output the results with a DFM RDB operation.

```
via_coverage {
    DFM RDB via_coverage eqdrc.rdb CHECKNAME "%_l_"
        CELL SPACE ALL CELLS
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Results

Results are shown in the following figure.



The lower right enclosure area has a portion of a fifth via within the enclosure, which causes the vias_extra property value to be 1. The via_area_small property value is non-zero only when the actual via area is smaller than expected, so its value is zero.

The vias are not centered in the lower right enclosure, but this check does consider via placement. Other checks, such as enclosure, width, and spacing rules, are expected to catch errors not considered by the via count and area checks demonstrated here.

Related Topics

Introduction to DFM Property [Calibre Verification User's Manual]

Measurement Functions [Standard Verification Rule Format (SVRF) Manual]

Math Operators [Standard Verification Rule Format (SVRF) Manual]

Introduction to DFM RDB [Calibre Verification User's Manual]

Supported Math Functions in DFM Expressions [Standard Verification Rule Format (SVRF) Manual]

Non-Persistent DFM Properties [Standard Verification Rule Format (SVRF) Manual]

Expression Chains [Standard Verification Rule Format (SVRF) Manual]

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Device and Routing Checks

This section includes checks related to routing layers and devices.

Table 2-2. Device-Level and Routing Check Solutions

Name	Description	SVRF Elements
Checking Device Orientation	Checks that all devices on a given marker layer are oriented in the same direction.	AND, DFM Property, DFM RDB, Grow, NOT
Checking Gate Width Dependent Via Count	Checks that via transitions have a minimum number of redundant vias to handle the current for the total gate width being driven by a particular metal line.	AND, Connect, DFM Property, DFM RDB, Inside Edge, Interact, Length
Writing a Wide Metal Spacing Check	Shows two options for writing wide metal spacing checks using traditional DRC and eqDRC.	External, DFM Property, DFM RDB, Internal
Finding the Length of a Net Path Connected to Gate	Finds an approximate path length of nets connected to MOS gates on the first two layers of metal interconnect. Two methods are presented and compared.	AND, Connect, DFM Copy, DFM Property, Interact, Length, Net Area Ratio, NOT, DFM RDB.
Checking Unsilicided Resistors	Checks multiple conditions for an unsilicided poly or active resistor.	Inside Edge, DFM Property, DFM RDB, Length,

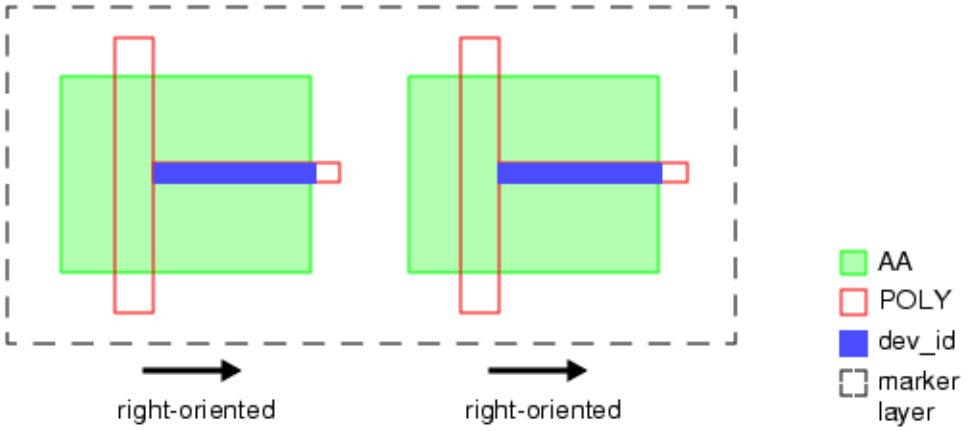
Checking Device Orientation

You can use marker layers to identify and check device orientation. This rule checks that all devices within the same marker layer shape have the same orientation.

The following figure shows a valid marker layer, since both devices are right-oriented. This particular marker layer would fail if one of the devices were oriented in a different direction.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 2-7. Sample Layout With Device Markers



This check can be implemented using several Grow operations and a single DFM Property operation.

Try It!	Calibre Advanced DRC (eqDRC) Tutorial and Example Kit
	Go to this page on Support Center to download the complete eKit. This example is in the device_routing_checks/device_orientation example directory.

Procedure

1. Derive the layer used to identify the device orientation.

```
dev_id = (INT POLY < 0.1 REGION) AND AA
```

Note

In this case, the REGION keyword produces rectangular polygons. In general, processing the output of the REGION keyword when skew polygons are generated can negatively impact performance.

2. Derive the layers used to check the device orientation.

   There are four possible orientations. In the previous figure (Figure 2-7), the dev_id layers are surrounded on three sides by active area (AA). To check the device orientation in this case, use four Grow operations:

```
d_l = (GROW dev_id LEFT BY 0.01) NOT AA
d_r = (GROW dev_id RIGHT BY 0.01) NOT AA
d_t = (GROW dev_id TOP BY 0.01) NOT AA
d_b = (GROW dev_id BOTTOM BY 0.01) NOT AA
```

   Each Grow operation expands the dev_id layer by a small amount in a specified direction. If the resultant polygon is inside the AA region, it is eliminated with the NOT operation. For the example layout shown in Figure 2-7, only layer d_r contains shapes.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

3. Use a DFM Property operation with a non-persistent property expression to identify marker layers that contain devices oriented in more than one direction:

```
wrong_orient = DFM PROPERTY marker d_l d_r d_t d_b OVERLAP ABUT ALSO
[- = !!COUNT(d_l) + !!COUNT(d_r) + !!COUNT(d_t) + !!COUNT(d_b)] > 1
```

   The !!COUNT function returns 0 if the count is zero, and 1 if the count is non-zero. If all devices on the marker layer are oriented in the same direction, the expression returns 1. If two or more devices are oriented in two or more directions on the marker layer, the expression returns a value greater than 1 and the expression does not pass the constraint.

4. Output the results with a DFM RDB operation.

```
wrong_orient {
     DFM RDB wrong_orient eqdrc.rdb CHECKNAME "%_l_"
         CELL SPACE ALL CELLS
}
```

Related Topics

Introduction to DFM Property [Calibre Verification User's Manual]

Measurement Functions [Standard Verification Rule Format (SVRF) Manual]

Math Operators [Standard Verification Rule Format (SVRF) Manual]

Introduction to DFM RDB [Calibre Verification User's Manual]

Supported Math Functions in DFM Expressions [Standard Verification Rule Format (SVRF) Manual]

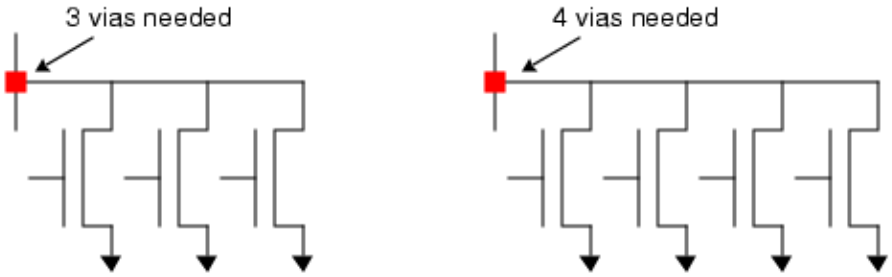
Grow [Standard Verification Rule Format (SVRF) Manual]

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Checking Gate Width Dependent Via Count

You can identify via transitions that have an insufficient number of redundant vias to handle the current for the total gate width being driven by a particular metal line. You can do this by writing a rule check that verifies there is at least one via per unit gate width.

Figure 2-8. Gate Width Dependent Via Count



There are several different ways to approach the problem. Performance between these approaches can vary based on factors including design size and hierarchy scheme, reuse of derived layers and Connect statements elsewhere in the rule file, and whether incremental connectivity is enabled. You can test all three implementations and choose the one with the best performance for your design data.

Checking Gate Width Dependent Via Count With DFM Property CONNECTED . . . 32
Checking Gate Width Dependent Via Count With DFM Property NODAL and Expand Edge. 35
Checking Gate Width Dependent Via Count With DFM Property NODAL and DFM Stamp 37

Checking Gate Width Dependent Via Count With DFM Property CONNECTED

This example uses DFM Property CONNECTED to identify via transitions that do not have enough redundant vias to handle the current for the total gate width being driven by a particular metal line.

Try It!	**Calibre Advanced DRC (eqDRC) Tutorial and Example Kit**
	Go to this page on Support Center to download the complete eKit.
	This example is in the device_routing_checks/gate_width_via_count example directory.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Procedure

1. Define the gate width value per via variable width_per_via:

   ```
   VARIABLE width_per_via 1
   ```

   The total gate width divided by width_per_via gives the fractional number of vias required for the total gate width.

2. Derive the sd (source-drain) and via transition layers and establish connectivity:

   ```
   sd = AA NOT POLY
   viatrans = (AND M1 M2) INTERACT VIA1

   CONNECT M1 M2 BY VIA1
   ```

3. Derive a layer that contains the edges of each gate:

   ```
   gateedges = INSIDE EDGE POLY AA
   ```

4. For each source-drain region, measure the width of the gate that the region overlaps. The following DFM Property operation uses sd as the primary layer and gateedges as the secondary clustering layer. The gatewidth property uses the LENGTH function to get the width of the gate for each source-drain region:

   ```
   sdprops = DFM PROPERTY sd gateedges OVERLAP ABUT ALSO MULTI
                 [gatewidth = LENGTH(gateedges)]
   CONNECT M1 sdprops BY CON
   ```

   Connect the sdprops layer to the M1 layer for use in the next step.

5. Sum the gate widths for each source-drain region that connects to an M1 geometry. Do this by using a DFM Property operation with the CONNECTED keyword, M1 as the primary layer, and sdprops as the secondary layer. The property sum_gatewidth is the sum of the gatewidth property over all sdprops shapes connected to the M1 shape.

   ```
   m1props = DFM PROPERTY M1 sdprops OVERLAP CONNECTED MULTI
             [sum_gatewidth = PROPERTY(sdprops,gatewidth)]
   ```

6. Verify that the required number of vias are present and define a property vias_to_add that displays how many vias must be added to pass the check:

   These properties are calculated:

   - “+vias_needed” — The fractional number of vias that are required for the total gate width (sum_gatewidth/width_per_via). This is a named non-persistent property, meaning it is not saved to the output layer, but the value can be referenced later in the same expression using PROPERTY_REF.
   - error_ratio — The required number of vias divided by the actual number of vias. A result is output if this value is greater than 1.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- vias_to_add — The number of vias that must be added to pass the check. The value is found by rounding the value of “+vias_needed” up to the nearest integer, then subtracting the actual number of vias.
- vias_present — The number of vias that overlap the viatrans shape.

```
eqdrc_err_con = DFM PROPERTY viatrans m1props VIA1 OVERLAP MULTI
    // fractional number of vias needed
    ["+vias_needed" = PROPERTY(m1props,sum_gatewidth)/width_per_via]
    [error_ratio = PROPERTY_REF("+vias_needed")/COUNT(VIA1) ] > 1
    [vias_to_add = CEIL(PROPERTY_REF("+vias_needed"))-COUNT(V1)]
    [vias_present = COUNT(VIA1)]
```

For efficiency, the sum_gatewidth property is not saved to the output layer. However, it can be calculated as error_ratio×vias_present×width_per_via.

7. Output the result to the DFM RDB file named *eqdrc_connected.rdb*. If desired, also output intermediate derived layers to the same file.

```
// Set default options for DFM RDB operations
DFM DEFAULTS RDB CELL SPACE ALL CELLS CHECKNAME "%_l_"
eqDRC_result {
    DFM RDB eqdrc_err_con eqdrc_connected.rdb
      COMMENT "Via Count per Gate Width Rule"
      COMMENT "DFM PROPERTY CONNECTED method"
      COMMENT "error_ratio = vias_needed/vias_present"
}
// Comment out to include debug output
DRC UNSELECT CHECK eqdrc_debug
eqdrc_debug {
    DFM RDB gateedges eqdrc_connected.rdb
    DFM RDB m1props eqdrc_connected.rdb
}
```

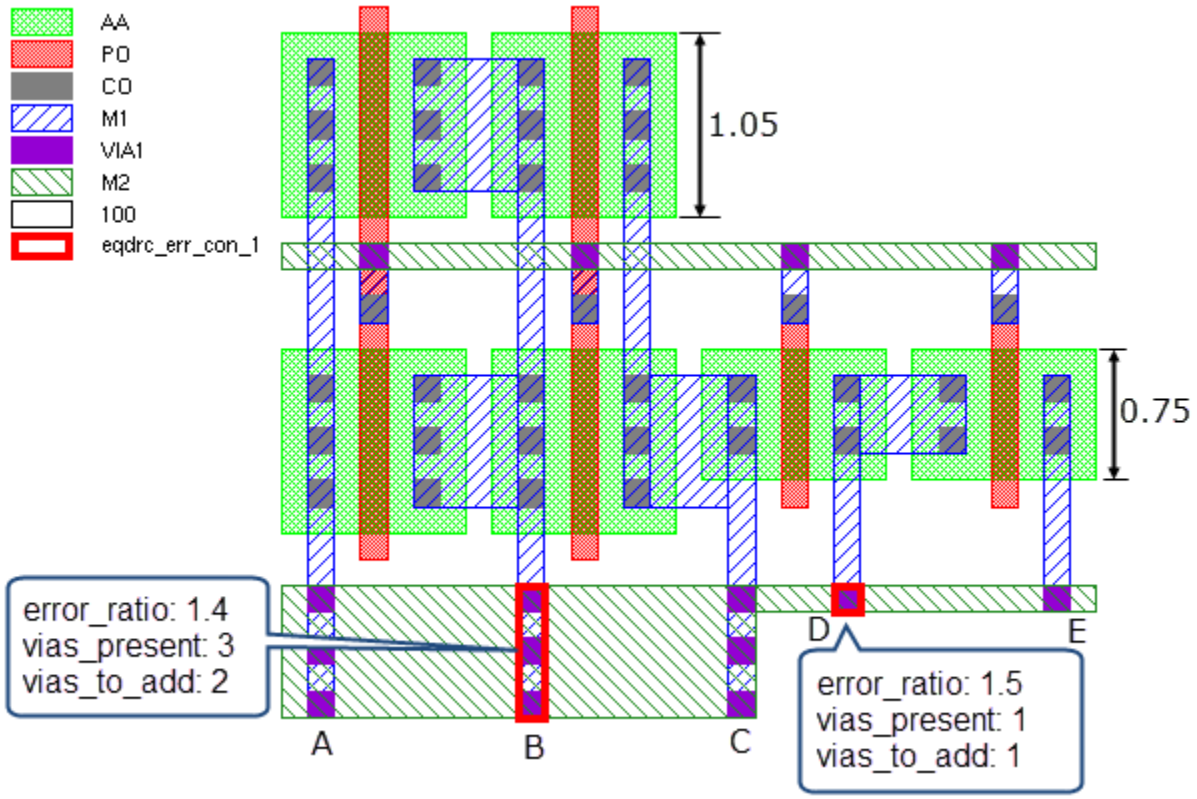
Results

Figure 2-9 shows output in Calibre DESIGNrev for a sample design. Several M1 lines are shown. The M1 lines that overlap source-drain regions are labeled A, B, C, D, and E. The via transitions for lines B and D are highlighted as errors because there are not enough vias for the total gate width.

Line B has a total gate width of 4.2 user units and 3 vias present. The gate width per via value (width_per_via) is 1, meaning 5 vias are needed (4.2 rounded up to the nearest integer). The property vias_to_add is 2, as expected. Line D has a total gate width of 1.5, meaning 2 vias are needed.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 2-9. Gate Width Dependent Via Count Output



Line A has a total gate width of 2.1 and passes the check. It requires 3 vias, which are present. Line C has a total gate width of 2.85 and passes the check with 3 vias present. Line E has a total gate width of 0.75 and passes the check with 1 via present.

Related Topics

Introduction to DFM Property [Calibre Verification User's Manual]

Non-Persistent DFM Properties [Standard Verification Rule Format (SVRF) Manual]

Measurement Functions [Standard Verification Rule Format (SVRF) Manual]

Checking Gate Width Dependent Via Count With DFM Property NODAL and Expand Edge

This example uses Expand Edge and DFM Property NODAL to identify via transitions that have an insufficient number of redundant vias to handle the current for the total gate width being driven by a particular metal line.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Try It!	**Calibre Advanced DRC (eqDRC) Tutorial and Example Kit**
	Go to this page on Support Center to download the complete eKit.
	This example is in the device_routing_checks/gate_width_via_count example directory.

Procedure

1. Define the gate width value per via variable width_per_via:

```
VARIABLE width_per_via 1
```

   The total gate width divided by width_per_via gives the fractional number of vias required for the total gate width.

2. Derive the sd (source-drain) and via transition layers and establish connectivity.

```
sd = AA NOT POLY
viatrans = (M1 AND M2) INTERACT VIA1

CONNECT M1 sd BY CON
CONNECT viatrans M1 BY VIA1
```

3. Derive a layer that contains the edges of each gate:

```
gateedges = INSIDE EDGE POLY AA
```

4. Expand the edges of the gate layer into the source-drain regions. Attach a property to the expanded layer that contains the gate width. Connect the expanded layer to the sd layer.

```
gateedgeP = EXPAND EDGE gateedges OUTSIDE BY 0.01

gateprops = DFM PROPERTY gateedgeP gateedges OVERLAP ABUT ALSO
                [gatewidth = LENGTH(gateedges)]

CONNECT gateprops sd
```

5. For each via transition, sum the gate widths for source-drain regions on the same net. Assign this value to the sum_gatewidth property.

```
total_gate = DFM PROPERTY viatrans gateprops NODAL MULTI
                 [sum_gatewidth = PROPERTY(gateprops,gatewidth)]
```

6. Verify that the required number of vias are present and define a property vias_to_add that displays how many vias must be added to pass the check:

```
eqdrc_err_EE = DFM PROPERTY total_gate VIA1 OVERLAP
   // fractional number of vias needed
   ["+vias_needed" = PROPERTY(total_gate,sum_gatewidth)/width_per_via]
   [error_ratio = PROPERTY_REF("+vias_needed")/COUNT(VIA1) ] > 1
   [vias_to_add = CEIL(PROPERTY_REF("+vias_needed"))-COUNT(VIA1)]
   [vias_present = COUNT(VIA1)]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

See "Checking Gate Width Dependent Via Count With DFM Property CONNECTED" on page 32 for more discussion of the calculated properties.

7. Output the result to the DFM RDB file named *eqdrc_expandEdge.rdb*. If desired, also output intermediate derived layers to the same file.

```
// Set default options for DFM RDB operations
DFM DEFAULTS RDB CELL SPACE ALL CELLS CHECKNAME "%_l_"
eqDRC_result {
    DFM RDB eqdrc_err_EE eqdrc_expandEdge.rdb
      COMMENT "Via Count per Gate Width Rule"
      COMMENT "EXPAND EDGE and DFM PROPERTY NODAL method"
      COMMENT "error_ratio = vias_needed/vias_present"
}
// Comment out to include debug output
DRC UNSELECT CHECK eqdrc_debug
eqdrc_debug {
    DFM RDB gateedges eqdrc_expandEdge.rdb
    DFM RDB total_gate eqdrc_expandEdge.rdb
}
```

Checking Gate Width Dependent Via Count With DFM Property NODAL and DFM Stamp

This example uses DFM Stamp and DFM Property NODAL to identify via transitions that have an insufficient number of redundant vias to handle the current for the total gate width being driven by a particular metal line.

Try It!	**Calibre Advanced DRC (eqDRC) Tutorial and Example Kit**
	Go to this page on Support Center to download the complete eKit.
	This example is in the device_routing_checks/gate_width_via_count example directory.

Procedure

1. Define the gate width value per via variable width_per_via:

```
VARIABLE width_per_via 1
```

The total gate width divided by width_per_via gives the fractional number of vias required for the total gate width.

2. Derive the sd (source-drain) and via transition layers and establish connectivity.

```
sd = AA NOT POLY
viatrans = (M1 AND M2) INTERACT VIA1

CONNECT M1 sd BY CON
CONNECT viatrans M1 BY VIA1
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

3. Derive a layer that contains the edges of each gate:

```
gateedges = INSIDE EDGE POLY AA
```

4. Derive a layer sd_n with a property “N” that contains the net ID for each sd geometry. Derive another layer gateedges_props with properties containing the gate width and the net ID values from layer sd_n. Finally, assign the node numbers to layer new_gateedges from the net ID property “N” on layer gateedges_props.

```
sd_n = DFM PROPERTY sd [N = NETID(sd)]

gateedges_props = DFM PROPERTY gateedges sd_n
    OVERLAP ABUT ALSO MULTI
    [gatewidth = LENGTH(gateedges)]
    [N = NETPROPERTY(sd_n, N, 1)]

new_gateedges = DFM STAMP gateedges_props BY "N"
```

DFM Stamp copies the gatewidth property from gateedges_props to the output layer.

5. Derive a layer total_gate with a property that contains the total gate width per node for each viatrans geometry.

```
total_gate = DFM PROPERTY viatrans new_gateedges NODAL MULTI
                [sum_gatewidth = PROPERTY(new_gateedges,gatewidth)]
```

6. Verify that the required number of vias are present and define a property vias_to_add that displays how many vias must be added to pass the check:

```
eqdrc_err_stamp = DFM PROPERTY total_gate VIA1 OVERLAP
   // fractional number of vias needed
   ["+vias_needed" = PROPERTY(total_gate,sum_gatewidth)/width_per_via]
   [error_ratio = PROPERTY_REF("+vias_needed")/COUNT(VIA1) ] > 1
   [vias_to_add = CEIL(PROPERTY_REF("+vias_needed"))-COUNT(VIA1)]
   [vias_present = COUNT(VIA1)]
```

See “Checking Gate Width Dependent Via Count With DFM Property CONNECTED” on page 32 for more discussion of the calculated properties.

7. Output the result to the DFM RDB file named *eqdrc_stamp.rdb*. If desired, also output intermediate derived layers to the same file.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// Set default options for DFM RDB operations
DFM DEFAULTS RDB CELL SPACE ALL CELLS CHECKNAME "%_l_"
eqDRC_result {
    DFM RDB eqdrc_err_stamp "eqdrc_stamp.rdb"
        COMMENT "Via Count per Gate Width Rule "
        COMMENT "DFM STAMP and DFM PROPERTY NODAL method"
        COMMENT "error_ratio = vias_needed/vias_present"
}
// uncomment to include debug output
DRC UNSELECT CHECK eqdrc_debug
eqdrc_debug {
    DFM RDB new_gateedges "eqdrc_stamp.rdb"
    DFM RDB total_gate "eqdrc_stamp.rdb"
}
```

Related Topics

Introduction to DFM Property [Calibre Verification User's Manual]

Measurement Functions [Standard Verification Rule Format (SVRF) Manual]

Math Operators [Standard Verification Rule Format (SVRF) Manual]

Introduction to DFM RDB [Calibre Verification User's Manual]

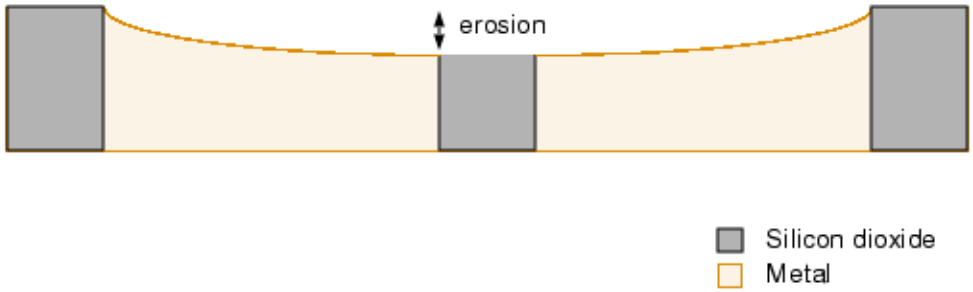
Supported Math Functions in DFM Expressions [Standard Verification Rule Format (SVRF) Manual]

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Writing a Wide Metal Spacing Check

Wide metal spacing refers to a layout challenge where wide metal paths separated by small areas of silicon dioxide can lead to performance and yield problems. Since metal is softer than oxide, it is removed more easily during traditional Chemical Mechanical Polishing (CMP) processes,6 and can lead to the erosion effects.

Figure 2-10. Erosion Effects



If the metal paths (or bands) are too wide relative to the area of oxide between them, there is increased susceptibility to erosion because the oxide does not create enough resistance against the polisher and the two metal areas behave as a single large area of metal. Increased erosion can lead to undesirable changes in resistance and capacitance, and can affect layers above due to changes in topology. Layout designers must take care to avoid this problem by ensuring a minimal separation distance for a number of metal widths.

For a more detailed discussion of implementing wide metal spacing checks using eqDRC, refer to "Wide Metal Spacing Rules" in the *Calibre Verification User's Manual*.

Writing Wide Metal Spacing Checks With Traditional DRC 40
Writing Wide Metal Spacing Checks Using a Continuous Function 42
Writing Wide Metal Spacing Checks Using DFM Function 45

Writing Wide Metal Spacing Checks With Traditional DRC

Commonly, addressing wide metal spacing checks involves a series of single-dimensional design rules. A minimum space value is associated with each range of widths for a single band and both bands.

Conceptually, the spacing rules are described as follows:

```
Rule1: 0.09 minimum M1 Space if both bands >= 0.09 wide
Rule2: 0.25 minimum M1 Space if one band   >= 0.36 wide
Rule3: 0.30 minimum M1 Space if both bands >= 0.36 wide
Rule4: 0.40 minimum M1 Space if one band   >= 1.44 wide
Rule5: 0.50 minimum M1 Space if both bands >= 1.44 wide
Rule6: 0.55 minimum M1 Space if one band   >= 4.50 wide
Rule7: 0.60 minimum M1 Space if both bands >= 4.50 wide
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Try It!	**Calibre Advanced DRC (eqDRC) Tutorial and Example Kit**
	Go to this page on Support Center to download the complete eKit.
	This example is in the device_routing_checks/wide_metal_spacing example directory.

Procedure

Write the wide-metal spacing rules to check a single band and both bands:

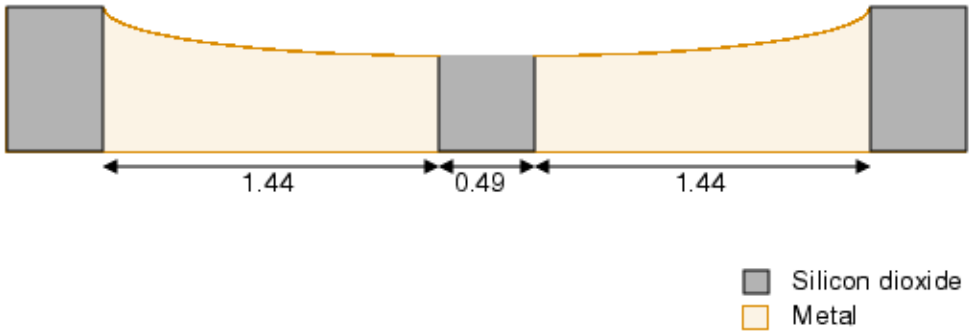
```
wide009 = Metal WITH WIDTH < 0.36
wide036 = Metal WITH WIDTH >= 0.36 < 1.44
all144 = wide009 OR wide036
wide144 = Metal WITH WIDTH >= 1.44 < 4.50
all450 = all144 OR wide144
wide450 = Metal NOT WITH WIDTH < 4.50

rule1_err { EXT wide009 < 0.09 OPPOSITE }
rule2_err { EXT wide036 wide009 < 0.25 OPPOSITE }
rule3_err { EXT wide036 < 0.30 OPPOSITE }
rule4_err { EXT wide144 all144 < 0.40 OPPOSITE }
rule5_err { EXT wide144 < 0.50 OPPOSITE }
rule6_err { EXT wide450 all450 < 0.55 OPPOSITE }
rule7_err { EXT wide450 < 0.60 OPPOSITE }
```

Results

The following analysis shows that the effectiveness of this approach can limit the erosion effects.

The following figure shows a case where both bands (or wires) are 1.44 units wide, and the separation distance is 0.49 units. This design fails rule 5.

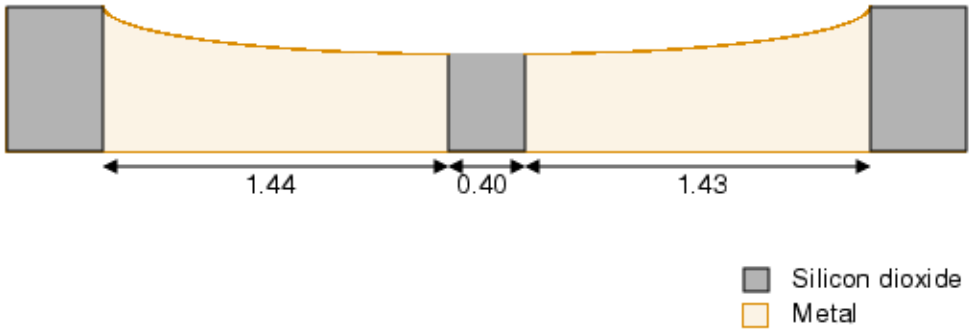
Figure 2-11. Case 1



The following figure shows a case where the bands are 1.44 and 1.43 units wide, and the separation distance is 0.4 units. Rule 4 now applies instead of rule 5, and the design passes.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

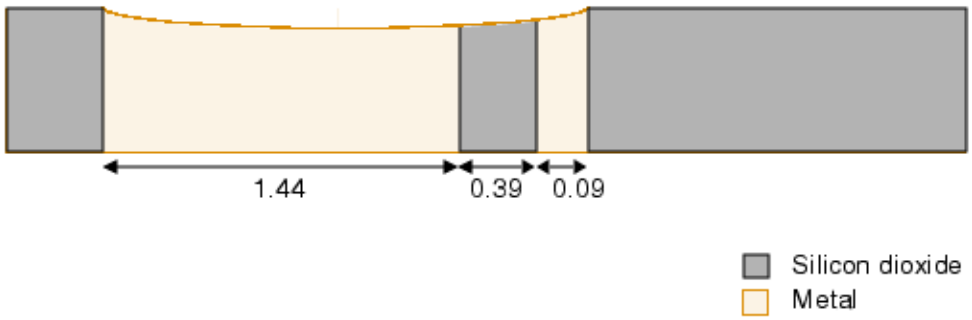
Figure 2-12. Case 2



Considering erosion susceptibility, the failing case shown in Figure 2-11 is actually better than the passing case shown in Figure 2-12. In the passing case, since the oxide area is smaller, more erosion will occur.

In Figure 2-13, one band has a width of 1.44 units, but the other band is only 0.09 units wide. Since one band is 1.44 units, the design fails rule 4 and the designer must increase the separation distance. Again, from a standpoint of susceptibility to erosion, Figure 2-13 is much better than Figure 2-12.

Figure 2-13. Case 3



Related Topics

Wide Metal Spacing Check [Calibre Verification User's Manual]

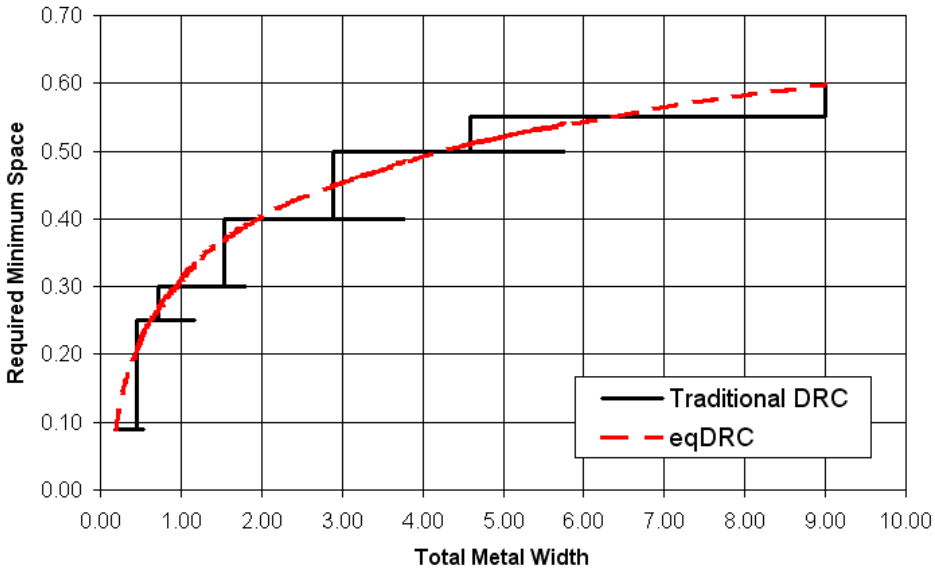
Writing Wide Metal Spacing Checks Using a Continuous Function

You can write a wide metal spacing check by implementing a continuous function in terms of the width and space values.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

To do this, find a curve of best fit to the discrete values defined by the traditional DRC rules, as shown in the following figure:

Figure 2-14. Traditional DRC Versus eqDRC



Note

There are intervals that extend to the right of the stair step pattern for each minimum space value on the traditional DRC plot. This is caused by how the traditional DRC rules are defined. There is no upper constraint on width for one band or both bands, only that the widths must be greater than or equal to a minimum value. Because of this, for each space value, the total metal width can continue to increase without violating any of the traditional DRC rules.

An equation can be obtained from the red curve in terms of the two variables—the separation distance (*space*) and the total width of the bands (*width1* plus *width2*). The equation can be put in the form of an error condition, where the check fails if the result is greater than 1:

$$\frac{0.09}{space \cdot \ln(2)} \cdot \ln\left(\frac{totalwidth}{0.09}\right) > 1$$

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The equation can be manipulated to find the change in spacing or total width needed to pass the check:

$$\Delta space = \frac{0.09}{\ln(2)} \cdot \ln\left(\frac{totalwidth}{0.09}\right) - space$$

$$\Delta totalwidth = \left(0.09 \cdot e^{\left(\frac{space \cdot \ln(2)}{0.09}\right)}\right) - totalwidth$$

For more details about how these equation are derived, see "Wide Metal Spacing Rules" in the *Calibre Verification User's Manual*.

Try It!	**Calibre Advanced DRC (eqDRC) Tutorial and Example Kit**
	Go to this page on Support Center to download the complete eKit.
	This example is in the device_routing_checks/wide_metal_spacing example directory.

Procedure

1. Create a derived error layer where the edge clusters mark the separation distance between the bands:

   ```
   m_space = DFM SPACE Metal <= 0.60 BY EXT
   ```

   This layer is used as the primary input layer to the DFM Property operation in Step 3.

2. Create a derived error layer where the edge clusters mark the width of each band:

   ```
   m_width = DFM SPACE Metal <= 5.00 BY INT
   ```

3. Create a derived error layer eqdrc_equation with three properties attached: error_value, increase_space_by, and reduce_sum_width_by:

```
eqdrc_equation = DFM PROPERTY m_space m_width OVERLAP ABUT ALSO
                 MULTI SPLIT ALL
[error_value = 0.09/(EW(m_space)*LOG(2)) * log(EW(m_width)/0.09)] > 1
[increase_space_by = (0.09/LOG(2))*(EW(m_width)/0.09)-EW(m_space)]
[reduce_sum_width_by = EW(m_width)-(0.09*EXP((EW(m_space)*LOG(2))/0.09))]
```

   The eqdrc_equation layer is the subset of the primary input layer m_space for which the property constraints are met. The EW measurement function returns the separation distance of the input error layer. Because m_width is a secondary layer, the EW values are automatically summed over secondary geometries in the cluster, so that EW(m_width) returns the total width of the two metal bands that abut the m_space error cluster.

4. Output the eqdrc_err layer to a DFM RDB file named *eqdrc.rdb*.

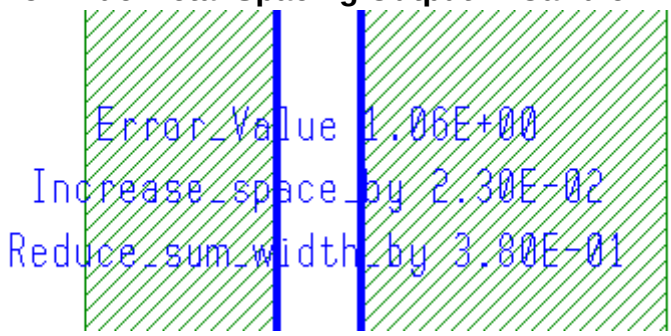
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
eqDRC_rule_eq {
    DFM RDB eqdrc_equation eqdrc.rdb CHECKNAME "%_l_"
       ALL CELLS CELL SPACE
       COMMENT "Wide metal spacing Rule: Use the property values to
       determine the error magnitude and adjustments needed to
       fix the violation"
}
```

Results

The following figure shows a result highlighted in Calibre DESIGNrev for a pair of failing metal bands.

Figure 2-15. Wide Metal Spacing Output in Calibre DESIGNrev



To fix the error, you can either increase the spacing by 2.3 units, or decrease the total width by 0.38 units, as indicated by the property values shown in the figure.

Related Topics

Introduction to DFM Property [Calibre Verification User's Manual]

Wide Metal Spacing Rules [Calibre Verification User's Manual]

Introduction to DFM RDB [Calibre Verification User's Manual]

Measurement Functions [Standard Verification Rule Format (SVRF) Manual]

Writing Wide Metal Spacing Checks Using DFM Function

You can use a DFM Function statement to simplify coding of traditional bucketed rules, such as those typically used for wide metal spacing checks.

This method offers similar advantages to the method described in "Writing Wide Metal Spacing Checks Using a Continuous Function" on page 42 (reduced code complexity and the ability to show correction values in a layout viewer), but it does not make use of a continuous function.

Try It!	**Calibre Advanced DRC (eqDRC) Tutorial and Example Kit**
	Go to this page on Support Center to download the complete eKit. This example is in the device_routing_checks/wide_metal_spacing example directory.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Procedure

1. Create a derived error layer where the edge clusters mark the separation distance between the bands:

```
m_space = DFM SPACE Metal <= 0.60 BY EXT
```

2. Create a derived error layer where the edge clusters mark the width of each band:

```
m_width = DFM SPACE Metal <= 5.00 BY INT
```

3. Define a DFM Function statement that accepts two metal width values as inputs and outputs the required spacing.

```
DFM FUNCTION [space_cal(NUMBER x, NUMBER y)
              (x < 0.36 && y < 0.36)
                ? 0.09 : (x < 1.44 && y < 0.36)
                ? 0.25 : (x < 1.44 && y < 1.44)
                ? 0.30 : (x < 4.50 && y < 1.44)
                ? 0.40 : (x < 4.50 && y < 4.50)
                ? 0.50 : (x < 4.50 && y < 1.44)
                ? 0.50 : (x >= 4.50 && y < 4.50)
                ? 0.55 : 0.60
              ]
```

4. Create a derived error layer eqdrc_bucket_fnct with attached properties that define the metal widths, required spacing, and correction values:

```
eqdrc_bucket_fnct = DFM PROPERTY m_space m_width OVERLAP MULTI SPLIT ALL
          [Short_width = MIN(EW(m_width))]
          [Long_width = MAX(EW(m_width))]
          [Space = MAX(EW(m_space))]
          [Increase_space_by =
            space_cal(PROPERTY_REF(Long_width),PROPERTY_REF(Short_width))
            - PROPERTY_REF(Space) ] > 0
```

   The property Increase_space_by calculates the correction value for the spacing by subtracting the actual spacing from the required spacing determined by the function space_cal.

5. Output the eqdrc_bucket_fnct layer to a DFM RDB file named *eqdrc.rdb*.

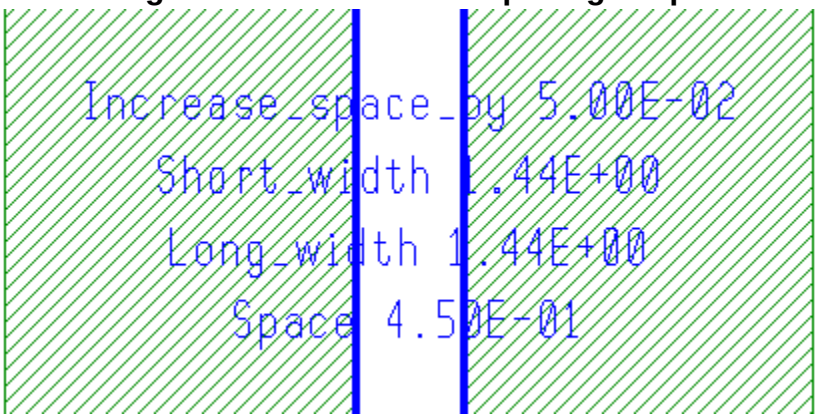
```
eqDRC_rule_dfm_function {
  DFM RDB eqdrc_err eqdrc.rdb CHECKNAME "%_l_"
      ALL CELLS CELL SPACE
      COMMENT "Wide metal spacing rule: Use the Increase_space_by "
      COMMENT "property value to determine the spacing adjustment "
      COMMENT "needed to fix the violation"
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Results

The following figure shows the output in Calibre DESIGNrev for a pair of failing metal bands:

Figure 2-16. Wide Metal Spacing Output



Related Topics

Introduction to DFM Property [Calibre Verification User's Manual]

Wide Metal Spacing Rules [Calibre Verification User's Manual]

Introduction to DFM RDB [Calibre Verification User's Manual]

Introduction to DFM Function [Calibre Verification User's Manual]

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Finding the Length of a Net Path Connected to Gate

The two rule checks in this section measure the path lengths of nets connected to MOS gates on the first two layers of metal interconnect.

Topic	Summary
Finding the Length of a Net Path Connected to Gate: Effective Length Method	An effective length of each polygon in the net path is estimated using an equation.
Finding the Length of a Net Path Connected to Gate: Centerline Method	The polygons that are part of the net are broken into line and corner segments, and the centerlines are found. The centerline length is summed to find the path length.
Comparison of Centerline and Effective Length Methods for Finding Path Length	The results of the two length estimates are compared for a selection of shapes.

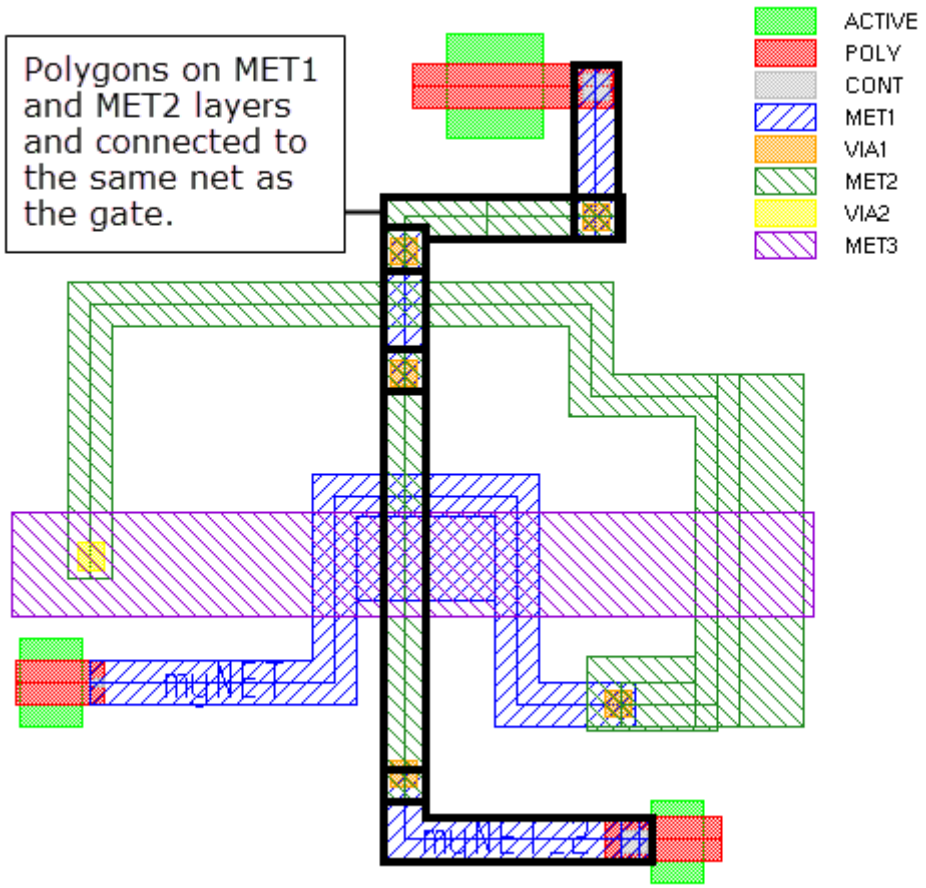
Finding the Length of a Net Path Connected to Gate: Effective Length Method

This check measures the path lengths of nets connected to MOS gates on the first two layers of metal interconnect. An effective length of each polygon in the path is estimated using an equation.

The following figure shows the polygons in the path connected to the top and bottom gates.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 2-17. Net Path Connected to Gate



The effective length of a polygon is given by this equation, where P is the polygon perimeter and A is the area:

$$\text{effectiveLength} = (P + \sqrt{P^2 - 16A})/4$$

Try It!	Calibre Advanced DRC (eqDRC) Tutorial and Example Kit
	Go to this page on Support Center to download the complete eKit. This example is in the device_routing_checks/gate_net_path_length/effective_length_method example directory.

Procedure

1. Make sure code similar to the following exists in your rules for connecting device gates up through the interconnect layer stack:

```
gate = POLY AND ACTIVE

CONNECT POLY gate
CONNECT MET1 POLY by CONT
CONNECT MET2 MET1 by VIA1
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

2. Add the following code as a macro to calculate the effective length for polygons on the input layer:

```
// effective length = (P + (sqrt(P^2 -16*A)))/4
// P = perimeter, A = area

DMACRO EffectiveLength layer_in layer_out {
       layer_out = DFM PROPERTY layer_in
           [leff = (PERIM(layer_in)
                    +SQRT(FMAX(PERIM(layer_in)*PERIM(layer_in)
                          - 16*AREA(layer_in),0)))/4 ]
           [Netname = NETID(layer_in)]
}
```

   The property leff contains the effective length. The netID of the input layer is attached for debugging purposes.

3. Find the MET1 and MET2 shapes on nets attached to gates,

```
// find metal layers on same net as gate polygon
M1 = NET AREA RATIO MET1 gate [!!AREA(gate)] == 1
M2 = NET AREA RATIO MET2 gate [!!AREA(gate)] == 1
```

4. Pass the M1 and M2 layers from the previous step to the macro from Step 2. Then sum the path length properties on the gate polygons and attach the sum as the property EffLength_M1M2.

```
// Call a macro to calculate effective lengths for layers M1 and M2.
// The output layers M1_len and M2_len have the property leff
// attached to each shape.
CMACRO EffectiveLength M1 M1_len
CMACRO EffectiveLength M2 M2_len

// Sum the lengths of M1 and M2 shapes on the same net as gate.
// Do not generate output if M2 shapes are not part of the path.
// Use this layer as an input for calculating path lengths on MET3
// if needed.

M1M2_gate =  DFM PROPERTY gate M1_len M2_len NODAL MULTI
          [ EffLength_M1M2 = (PROPERTY(M2_len,leff) == 0) ? 0 :
              PROPERTY(M1_len,leff) +  PROPERTY(M2_len,leff) ] > 0
          [ Netname = NETID(gate) ]
```

5. Output the results.

```
// Set default options for DFM RDB operations
DFM DEFAULTS RDB CELL SPACE ALL CELLS CHECKNAME "%_l_"
effective_path_length {
    DFM RDB M1M2_gate "eqdrc_effective_length.rdb"
}
```

6. If debug output is desired, comment out the DRC Unselect Check statement to run the debug check. The M1_len and M2_len layers have individual M1 and M2 shapes with attached properties leff and Netname.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
DRC UNSELECT CHECK path_length_debug
path_length_debug {
    DFM RDB M1_len "eqdrc_effective_length.rdb"
    DFM RDB M2_len "eqdrc_effective_length.rdb"
    DFM RDB M3_len "eqdrc_effective_length.rdb"
}
```

Results

Open the *eqdrc_effective_length.rdb* file in Calibre RVE to view the path length property EffLength_M1M2 for each gate. Choose **View** > **Result Options** > **Details** to view the results in table format.

	Check	EffLength_M1M2	Netname
1	M1M2_gate	5.34E+01	1 myNET
2	M1M2_gate	3.17E+01	2 myNET_2
3	M1M2_gate	3.17E+01	2 myNET_2

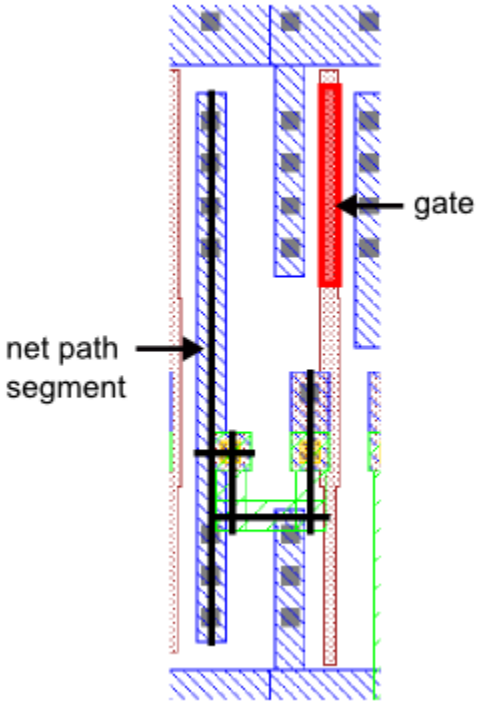
Finding the Length of a Net Path Connected to Gate: Centerline Method

This check measures the path lengths of nets connected to MOS gates along centerlines of complete polygons on the first two layers of metal interconnect. Length is not measured from via to via or via to contact, but along the entire lengths of polygons. Corners and jogs are handled reasonably so lengths are not counted twice at corners.

Figure 2-18 shows a MOS gate with its associated net path on MET1 and MET2 layers. The centerlines calculated by the layer derivation code are shown in black. The total lengths of the path segments on a net are attached to the associated gate polygons as properties.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 2-18. Net Path Connected to Gate



The rule file code in this example may require further refinement, depending on the behavior of jogs and corners on the metal layers; see "Comparison of Centerline and Effective Length Methods for Finding Path Length" on page 55 for examples of the derivation results with short stubs, non-uniform line widths, and angled edges. If net paths include more than two metal layers, you can add layers as needed by using the existing code as a template.

Try It!	**Calibre Advanced DRC (eqDRC) Tutorial and Example Kit**
	Go to this page on Support Center to download the complete eKit.
	This example is in the device_routing_checks/gate_net_path_length/centerline_method example directory.

Procedure

1. Ensure code similar to the following exists in your rules for connecting device gates up through the interconnect layer stack:

```
gate = POLY AND ACTIVE

CONNECT POLY gate
CONNECT MET1 POLY by CONT
CONNECT MET2 MET1 by VIA1
```

2. Define the following variables for the minimum length of a metal edge and for the maximum width of a metal polygon to be considered as part of a net path. Use values

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

suitable for your design. Edges shorter than cd_lineend are considered corners and handled separately.

```
// minimum edge length to consider for a line
VARIABLE cd_lineend 1
// maximum line width to consider
VARIABLE w_max 3
```

3. Add the following code as a macro for calculating polygon centerline lengths, summing the lengths, and placing the length properties on the output layers:

```
// gate is a MOS gate layer
// lay is a metal layer on a net
// lay_prop_nodal is an output layer with gate shapes
//     and attached properties
// lay_prop is an output layer with lay shapes and properties
DMACRO PathLength gate lay lay_prop_nodal lay_prop {

  // Get edges longer than the minimum
  lay_edges = LENGTH lay > cd_lineend

  // Get edges closer than max width of a wire
  lay_int = INT lay_edges <= w_max OPPOSITE

  // Get the centerlines between max width edges
  lay_e = DFM COPY lay_int CENTERLINE

  // Make regions out of max width edges
  lay_e_reg = DFM COPY lay_int REGION

  // Get corner regions of the wires
  lay_corner = lay NOT lay_e_reg

  // Get corner edges closer than max width of a wire
  lay_corner_int = INT lay_corner <= w_max OPPOSITE

  // Get the centerlines between max width edges on a corner
  lay_c = DFM COPY lay_corner_int CENTERLINE

  // Attach corner length properties to the input layer.
  // px=0 and py=0 are expression chains, or fall-through rules,
  //     in case of evaluation errors
  lay_c_prop = DFM PROPERTY lay lay_c OVERLAP MULTI
               [ px = SUM(LENGTHXP(lay_c)) ] [ px = 0 ]
               [ py = SUM(LENGTHYP(lay_c)) ] [ py = 0 ]
     [ pathlength_corner = PROPERTY_REF(px)/2 + PROPERTY_REF(py)/2 ]

  // Attach edge length properties to the input layer
  lay_e_prop = DFM PROPERTY lay lay_e OVERLAP MULTI
               [ pathlength_edge = SUM(LENGTH(lay_e)) ]
                   [ pathlength_edge = 0 ]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
  // Sum the corner and edge length properties on the input layer
  lay_prop = DFM PROPERTY lay lay_c_prop lay_e_prop OVERLAP
             ABUT ALSO MULTI
             [ plc = SUM(PROPERTY(lay_c_prop,pathlength_corner)) ]
             [ ple = PROPERTY(lay_e_prop,pathlength_edge) ]
             [ pl = PROPERTY_REF(plc) + PROPERTY_REF(ple) ]

  // Attach length and netID properties to gates
  lay_prop_nodal = DFM PROPERTY gate lay_prop NODAL MULTI
                   [ PathLength = PROPERTY(lay_prop,pl) ]
                   [ Netname = NETID(gate) ]
}
```

4. Add this code to identify the MET1 and MET2 shapes on nets attached to gates, and pass those layers to the macro from Step 3. Then sum the path length properties on the gate polygons and attach the sum as a new property PathLength_M1M2.

```
// Find metal layers on same net as gate polygon
M1 = NET AREA RATIO MET1 gate [!!AREA(gate)] == 1
M2 = NET AREA RATIO MET2 gate [!!AREA(gate)] == 1

// Call a macro to generate the derived layers PL_M1 and PL_M2,
// which contain the path length property PathLength for the metal
// layers connected to each gate
CMACRO PathLength gate M1 PL_M1
CMACRO PathLength gate M2 PL_M2

// Sum the lengths of M1 and M2 shapes on the same net as gate.
// Do not generate output if M2 shapes are not part of the path.
// Use this layer as an input for calculating path lengths on MET3
// if needed.
PL_M1M2 = DFM PROPERTY gate PL_M1 PL_M2 OVERLAP MULTI
    [ PathLength_M1M2 = (PROPERTY(PL_M2,PathLength) == 0) ? 0 :
      PROPERTY(PL_M1,PathLength) +  PROPERTY(PL_M2,PathLength) ] > 0
    [Netname = NETPROPERTY(PL_M1,"Netname",1)]
```

5. Output the results.

```
path_length {
    DFM RDB PL_M1M2 "eqdrc.rdb" CHECKNAME "%_l_"
       CELL SPACE ALL CELLS
}
```

6. If debug output is desired, comment out the DRC Unselect Check statement to run the PL1_dbg and centerlines_dbg checks. These checks can be used to verify the output of the preceding code.

```
DRC UNSELECT CHECK PL1_dbg centerlines_dbg

PL1_dbg {
@ show gates connected to M1 and M2.
@ show whole chains in nar_dbg.rdb.
  NET AREA RATIO PL_M1M2 M1 M2 > 0 RDB "nar_dbg.rdb"
    BY LAYER PL_M1M2 M1 M2
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// draw centerlines of polygons involved in net paths
DMACRO centerlines L {
  lay_edges   = LENGTH L > cd_lineend
  lay_int     = INT lay_edges <= w_max OPPOSITE
  DFM COPY lay_int CENTERLINE
  lay_e_reg   = DFM COPY lay_int REGION
  lay_corner = L NOT lay_e_reg
  lay_corner_int = INT lay_corner <= w_max OPPOSITE
  DFM COPY lay_corner_int CENTERLINE
}

centerlines_dbg {
@ show polygon centerlines, which are the basis for
@ calculating the path lengths
  CMACRO centerlines M1
  CMACRO centerlines M2
}
```

Results

Open the *eqdrc.rdb* file in Calibre RVE to view the path length property PathLength_M1M2 for each gate. Choose **View** > **Result Options** > **Details** to view the results in table format.

	Check	Netname	PathLength_M1M2
1	PL_M1M2	1 myNET	5.39E+01
2	PL_M1M2	2 myNET_2	3.14E+01
3	PL_M1M2	2 myNET_2	3.14E+01

If debug output was produced, open the *nar_dbg.rdb* database and the standard DRC results database. The standard DRC results database contains the output of the centerlines_dbg check.

Comparison of Centerline and Effective Length Methods for Finding Path Length

The path length of polygons in a net can be computed by two different methods: centerline method and effective length method.

- Finding the Length of a Net Path Connected to Gate: Effective Length Method

  Estimates the effective length of each polygon in the path using an equation.

- Finding the Length of a Net Path Connected to Gate: Centerline Method

  Finds the approximate centerlines of the polygons in the path and sums the total length of the centerlines. Edges that are less than a specified minimum edge length are considered corners and are handled separately, so that lengths are not counted twice at corners.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

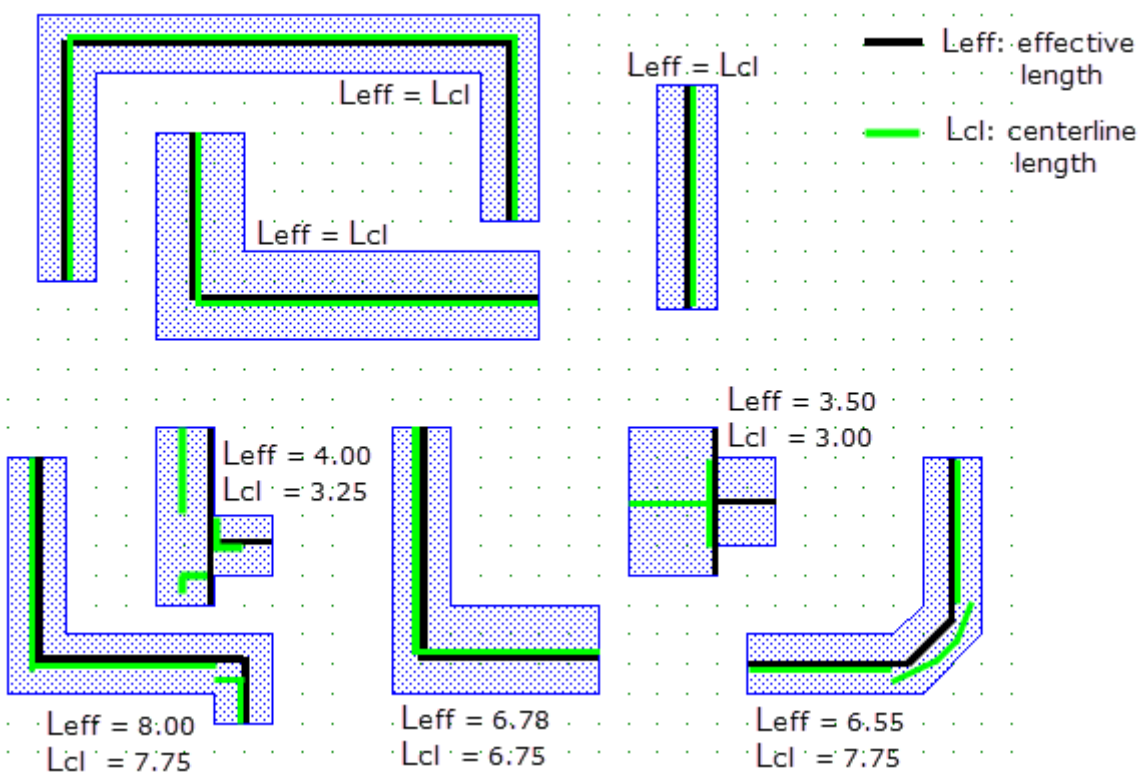
Table 2-3. Comparison of Path Length Computation Methods

Criteria	Centerline Method	Effective Length Method
Runtime	slow	fast
Applies to all designs?	Must specify an appropriate minimum edge length and line width	yes
Applies to all shapes?	Works best for Manhattan shapes with uniform width and no short stubs	yes

Figure 2-19 compares the length estimates for the two methods when applied to several shapes. The lines are approximations of the length for each method. For the centerline estimation method, a minimum edge length of 1.0 is used, and a maximum line width of 3.0. See the derivation in "Finding the Length of a Net Path Connected to Gate: Centerline Method" on page 51 to see how these values are used.

The three shapes in the top half of the figure are Manhattan shapes with a uniform width and no stubs, and the length estimates from the two methods are the same. The shapes in the bottom half of the figure have short stubs, non-uniform width, or angled edges, and the length estimates from the two methods are different. For the centerline method, a different value of the minimum edge length would give different results in some cases.

Figure 2-19. Comparison of Path Length Computation Methods



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

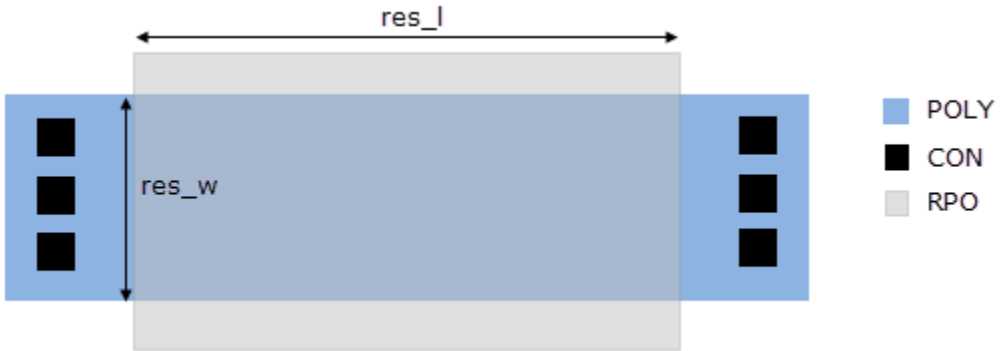
Checking Unsilicided Resistors

This example checks the width, length, and length to width ratio rules for an unsilicided resistor. Results, with attached properties, are only output for resistors that fail at least one of the rules.

An unsilicided resistor must pass the following checks:

- Width: res_w >=0.4 um
- Length: res_l >= 0.4 um
- Aspect ratio: res_l/res_w >=1

Figure 2-20. Unsilicided Resistor



These checks can be coded using a conditional expression combined with a non-persistent property that filters out resistors that pass all of the checks. Non-persistent properties, which are named "-", "+", or "+<*name*>", are not attached to the output layer. In this way, properties are attached only to resistors that fail one or more of the checks.

Try It!	**Calibre Advanced DRC (eqDRC) Tutorial and Example Kit**
	Go to this page on Support Center to download the complete eKit.
	This example is in the device_routing_checks/unsilicided_res example directory.

Procedure

1. Derive the length and width edges:

```
res_l = INSIDE EDGE POLY RPO
res_w = INSIDE EDGE RPO POLY
```

2. Filter out resistors that pass all of the checks.

```
bad_resistors = DFM PROPERTY RPO res_l res_w OVERLAP ABUT ALSO
                [- = ((LENGTH(res_w)/2 < 0.4) ||
                      (LENGTH(res_l)/2 < 0.4) ||
                      (LENGTH(res_l)/LENGTH(res_w) < 1.0)) ? 1 : 0
                ] == 1
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

A conditional expression is used to output geometries that fail one or more of the checks. The conditional expression reads as follows: if the first, second, or third condition is true, the DFM expression evaluates to 1, where each condition defines a failing check. Otherwise, the DFM expression evaluates to 0. Since the constraint on the DFM expression is “== 1”, the DFM Property operation outputs geometries on layer RPO that fail one or more of the checks.

3. Specify additional DFM expressions that attach properties to the failing resistors. The following lines can be appended to the DFM Property operation in Step 2:

```
[width = LENGTH(res_w)/2]
[length = LENGTH(res_l)/2]
[aspect = LENGTH(res_l)/LENGTH(res_w)]
```

   Since the geometries have already been filtered in Step 2, no constraints on the expressions are needed.

4. Output the results with a DFM RDB operation:

```
res_chk {
    DFM RDB bad_resistors eqdrc.rdb CHECKNAME "%_l_"
        CELL SPACE ALL CELLS
}
```

Related Topics

Introduction to DFM Property [Calibre Verification User's Manual]

Conditional Expressions [Standard Verification Rule Format (SVRF) Manual]

Non-Persistent DFM Properties [Standard Verification Rule Format (SVRF) Manual]

Math Operators [Standard Verification Rule Format (SVRF) Manual]

Introduction to DFM RDB [Calibre Verification User's Manual]

Measurement Functions [Standard Verification Rule Format (SVRF) Manual]

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Grid and Pitch Checks

This section includes grid and pitch checks.

Table 2-4. Grid and Pitch Check Solutions

Name	Description	SVRF Elements
Checking Polygon Centerline Alignment	Polygon centerlines must be aligned, where one polygon, such as a marker shape, overlaps another polygon from a different layer.	AND, NOT, Coincident Outside Edge, DFM Property, DFM DRB
Checking Equal Gate Pitch per Active Area	All gates in an active area region must have equal widths and must be equally spaced apart.	External, DFM Property, DFM DRB, Internal
Checking Gate Relative Pitch per Active Area	The centerline-to-centerline spacing of gates within the same active area must be spaced with an integer pitch value relative to each other.	AND, Variable, DFM Space, DFM Copy, DFM Property, DFM RDB
Checking Metal Routing Relative Pitch	The centerline-to-centerline spacing of routing wires must be spaced with an integer pitch value relative to each other.	Variable, DFM Space, DFM Copy, DFM Property, DFM RDB
Checking Contact Relative Pitch	The center points of contacts must be spaced with an integer pitch value relative to each other.	Variable, Extents, Grow, Offgrid, DFM RDB

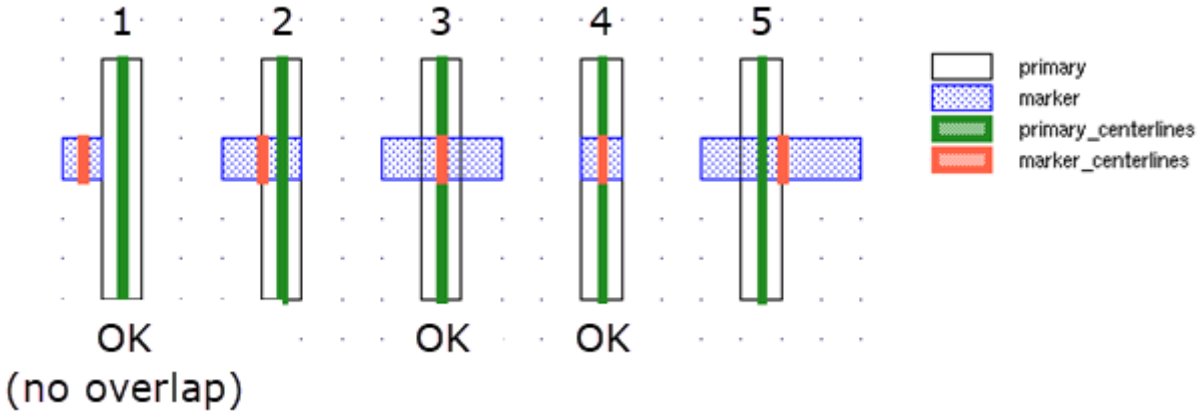
Checking Polygon Centerline Alignment

You can check that polygon centerlines are aligned, where one polygon, such as a marker shape, overlaps another polygon from a different layer.

In the following figure, primary layer shapes interact with marker layer shapes. The centerlines of the primary shape “width” dimension are the targets for alignment with centerlines of the marker shapes. Marker shapes whose centerlines do not align with the primary shapes are flagged, unless there is no overlap of the marker and primary shape.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 2-21. Check Centerline Alignment



The rule check compares areas of marker shape regions (marker NOT primary) on either side of the primary shapes. Additionally, the lengths of shared edge segments between primary shapes and the marker shape regions are checked. When the areas and segment lengths match, respectively, the marker shape is considered to be properly aligned. While checking the shared edge segments is not strictly necessary (especially if the shapes are guaranteed to be rectangular by some other check), it is possible to have equal areas when the centerlines do not align. Hence, checking the shared edge segment lengths provides a more robust check.

Case #4 in the preceding figure, where the primary shape completely overlaps the marker shape, is considered OK if two marker edges are inside coincident with the primary shape.

Try It!	**Calibre Advanced DRC (eqDRC) Tutorial and Example Kit**
	Go to this page on Support Center to download the complete eKit. This example is in the grid_pitch_checks/polygon_centerlines example directory.

Procedure

1. Derive the intersections of the primary and marker layers, the marker layer regions on either side of primary shapes, and the shared edges of the regions and the primary shapes:

```
pr_AND_mk = primary AND marker
mk_NOT_pr = marker NOT primary
p_COE_m_NOT_p = primary COIN OUTSIDE EDGE mk_NOT_pr
p_CIE_p_AND_m = primary COIN INSIDE EDGE pr_AND_mk
```

2. Use DFM Property to check that the marker and primary layer polygons which overlap have the same centerline.

   The check analyzes shapes interacting with pr_AND_mk:

   - Error if there are 1 or more than 2 mk_NOT_pr polygons (case #2)

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Else, error if there are 0 mk_NOT_pr polygons and the pr_AND_mk polygon does not have two COIN INSIDE edges with primary (case #4)
- Else, error if the lengths of the outside common edges of primary and mk_NOT_pr are unequal
- Else, error if the areas of mk_NOT_pr interacting with pr_AND_mk are unequal (case #5)

```
non_aligned = DFM PROPERTY pr_AND_mk
                           mk_NOT_pr p_COE_m_NOT_p p_CIE_p_AND_m
                           OVERLAP ABUT ALSO MULTI
    [ "+ne_area" = DRC_NE( MAX(AREA(mk_NOT_pr), 0) ,
                           MIN(AREA(mk_NOT_pr), 0) )]
    [ "+ne_len"  = DRC_NE( MAX(LENGTH(p_COE_m_NOT_p), 0) ,
                           MIN(LENGTH(p_COE_m_NOT_p), 0) ) ]
    [ - = ( (COUNT(mk_NOT_pr) > 2) || (COUNT(mk_NOT_pr) == 1 ) ) ? 0
:
          ( (COUNT(mk_NOT_pr) == 0) &&
            (COUNT(p_CIE_p_AND_m) != 2) ) ? 0 :
          ( PROPERTY_REF("+ne_len") > 0 ) ? 0 :
          ( PROPERTY_REF("+ne_area") > 0) ? 0 : 1 ] == 0
```

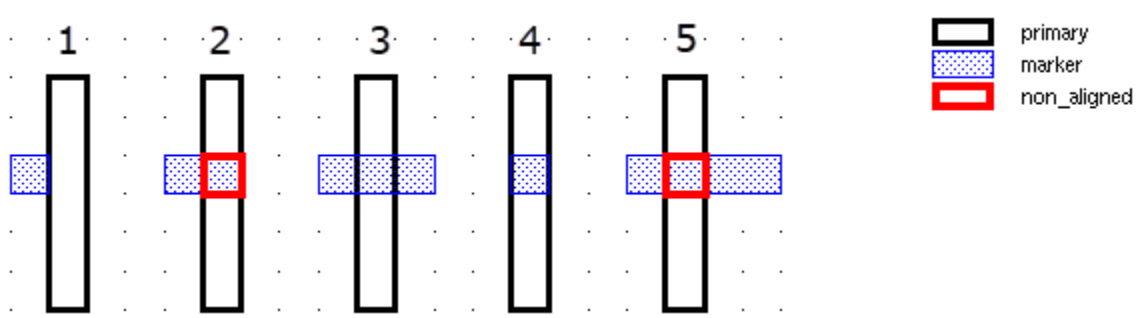
DRC_NE() returns 1.0 if its arguments are not equal within a built-in tolerance, and 0.0 otherwise. The MIN() and MAX() functions include a default value of 0, to prevent evaluation failure when the layer argument has no shapes in the cluster.

3. Output results using DFM RDB.

```
centerline_check {
   DFM RDB non_aligned eqdrc.rdb CHECKNAME "%_l_"
       CELL SPACE ALL CELLS
       COMMENT "Centerlines of marker and primary shape are not
                aligned"
}
```

Results

Compare the following results to Figure 2-21.



The two incorrect cases (#2 and #5) are flagged.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This check handles the cases pictured here and discussed in the text. It does not handle the case of two equal area mk_NOT_pr polygons on the same side of a pr_AND_mk shape, but that is probably an unusual configuration.

For cases where the marker shapes are not cut by the primary shapes, a similar check can be performed using REGION CENTERLINE with Internal or CENTERLINE with DFM Copy. The range of feature sizes must be known in advance for dimensional check operations, which is sometimes a challenge. In addition, if the measurement constraints are large, that can adversely affect performance when using operations like Internal. DFM Space is the better choice when measurement constraints are large.

Related Topics

Introduction to DFM Property [Calibre Verification User's Manual]

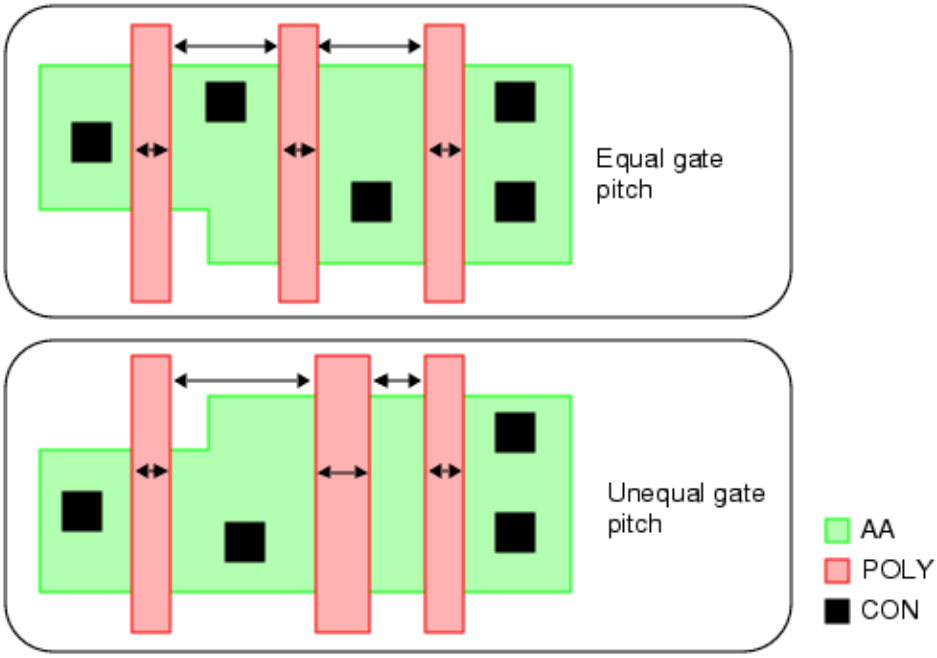
Non-Persistent DFM Properties [Standard Verification Rule Format (SVRF) Manual]

MAX [Standard Verification Rule Format (SVRF) Manual]

Checking Equal Gate Pitch per Active Area

You can identify active area (AA) regions where gates are not equally spaced apart, have unequal widths, or both.

Figure 2-22. Gate Pitch



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Try It!	**Calibre Advanced DRC (eqDRC) Tutorial and Example Kit**
	Go to this page on Support Center to download the complete eKit.
	This example is in the grid_pitch_checks/equal_gate_pitch_per_aa example directory.

Procedure

1. Derive gate spacing and width edge pairs:

```
VARIABLE gate_spc 0.4   // max gate to gate spacing
VARIABLE gate_wid 0.2   // max gate width
spc = EXT gate < gate_spc OPPOSITE
wid = INT gate < gate_wid OPPOSITE
```

2. Determine active areas with unequal pitch. To do this, use a DFM Property operation with AA as the primary layer, and spc and wid as the secondary clustering layers:

```
pitcherr_aa = DFM PROPERTY AA spc wid OVERLAP MULTI
                  [- = DRC_NE(MAX(EW(wid)), MIN(EW(wid)))
                       + DRC_NE(MAX(EW(spc)), MIN(EW(spc))) ] != 0
                  [spc_diff = MAX(EW(spc)) - MIN(EW(spc))]
                  [wid_diff = MAX(EW(wid)) - MIN(EW(wid))]
```

   The rule check works as follows:

   - Use the MAX and MIN measurement functions to find the maximum and minimum width and spacing in the active area AA.
   - Use the DRC_NE comparison function to determine if the maximum and minimum width and spacing values are not equal. DRC_NE returns 1.0 if the arguments are not equal within a built-in tolerance, and returns 0 otherwise.
   - If either DRC_NE function returns 1.0, output the result, which is the AA layer polygon.

3. (Optional) You can use a DFM Property Select Secondary operation to output the spc or wid error layer as a result rather than the AA layer polygon.

```
pitcherr_wid = DFM PROPERTY SELECT SECONDARY AA wid OVERLAP MULTI
                   [ - = DRC_NE(MAX(EW(wid)), MIN(EW(wid))) ] > 0
                   SELECT wid
                   [ wid = EW(wid)]

pitcherr_spc = DFM PROPERTY SELECT SECONDARY AA spc OVERLAP MULTI
                   [ - = DRC_NE(MAX(EW(spc)), MIN(EW(spc))) ] > 0
                   SELECT spc
                   [ spc = EW(spc)]
```

4. Output the results using DFM RDB.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
// Set default options for DFM RDB operations
DFM DEFAULTS RDB CELL SPACE ALL CELLS CHECKNAME "%_l_"
gate_pitch_active {
   DFM RDB pitcherr_aa advDRC.rdb
   DFM RDB pitcherr_wid advDRC.rdb
   DFM RDB pitcherr_spc advDRC.rdb
}
```

Related Topics

Introduction to DFM Property [Calibre Verification User's Manual]

Measurement Functions [Standard Verification Rule Format (SVRF) Manual]

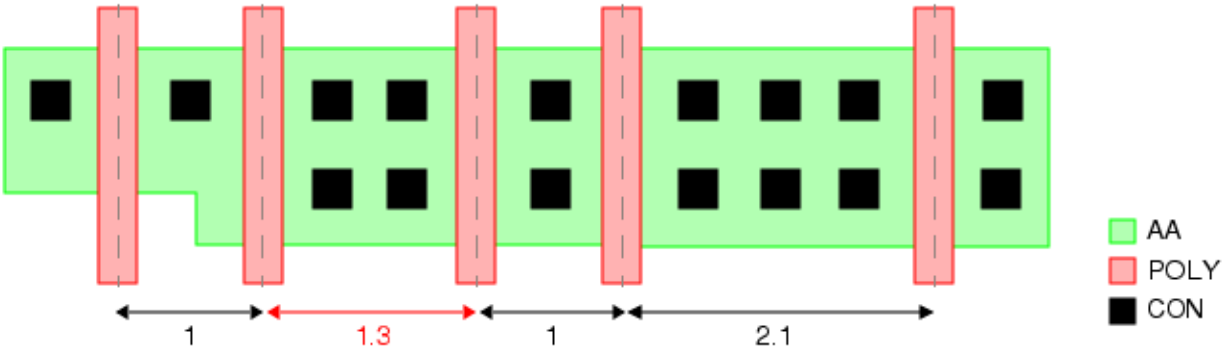
Introduction to DFM RDB [Calibre Verification User's Manual]

Math Operators [Calibre YieldAnalyzer and YieldEnhancer User's and Reference Manual]

Checking Gate Relative Pitch per Active Area

You can check that the centerline-to-centerline spacing of gates within the same active area are spaced with an integer pitch value relative to each other.

Figure 2-23. Centerline Gate Pitch



In the figure, the pitch is specified as 1 and the optical radius is specified as 2. The spacing value of 1.3 does not pass the check because it is not an integer multiple of 1. However, the spacing value of 2.1 passes because it is outside the optical radius value of 2.

Try It!	**Calibre Advanced DRC (eqDRC) Tutorial and Example Kit**
	Go to this page on Support Center to download the complete eKit.
	This example is in the grid_pitch_checks/gate_rel_pitch example directory.

Procedure

1. Derive the gate layer and define the pitch, optical radius, and maximum gate width values.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
gate = POLY AND AA

VARIABLE optradius 2
VARIABLE pitch 1
VARIABLE max_width 0.5
```

2. Generate centerlines for each gate.

```
gate_widths = DFM SPACE gate <= max_width BY INT HORIZONTAL
gate_cl = DFM COPY gate_widths CENTERLINE
```

3. Measure the distances between centerlines with DFM Space. Use the GRID keyword to output centerlines not on the given pitch. The ALL keyword is needed because DFM Copy with CENTERLINE produces edges that face arbitrary directions.

```
gate_cl_offpitch = DFM SPACE gate_cl <= optradius
                            ALL HORIZONTAL GRID pitch
```

4. Write a DFM Property operation that performs these functions:

   - Filter out spacing errors between different active area regions using a constraint of COUNT(AA) == 1 with a non-persistent property.
   - Perform the pitch calculations, The properties generated tell you by what value the spacing should be changed, the deviation from the correct pitch value as a percentage, the number of pitch values, and the spacing distance.

```
gate_pitch_err = DFM PROPERTY gate_cl_offpitch AA OVERLAP MULTI
    [- = COUNT(AA)] == 1
    [change_space_by = REMAINDER(EW(gate_cl_offpitch),pitch)] != 0
    [pct_off_pitch =
            ROUND(ABS(100*(PROPERTY_REF(change_space_by)/pitch)))]
    [num_pitches = ROUND(EW(gate_cl_offpitch)/pitch, 0.1)]
    [distance = EW(gate_cl_offpitch)]
```

5. Output the results using a DFM RDB operation.

```
gate_pitch {
  DFM RDB gate_pitch_err  "eqdrc.rdb" CHECKNAME "%_l_"
    CELL SPACE ALL CELLS
    COMMENT "Within each AA region, output gate centerlines "
    COMMENT "that are not on pitch of ^pitch"
}
```

Related Topics

Introduction to DFM Property [Calibre Verification User's Manual]

Measurement Functions [Standard Verification Rule Format (SVRF) Manual]

Supported Math Functions in DFM Expressions [Standard Verification Rule Format (SVRF) Manual]

Introduction to DFM RDB [Calibre Verification User's Manual]

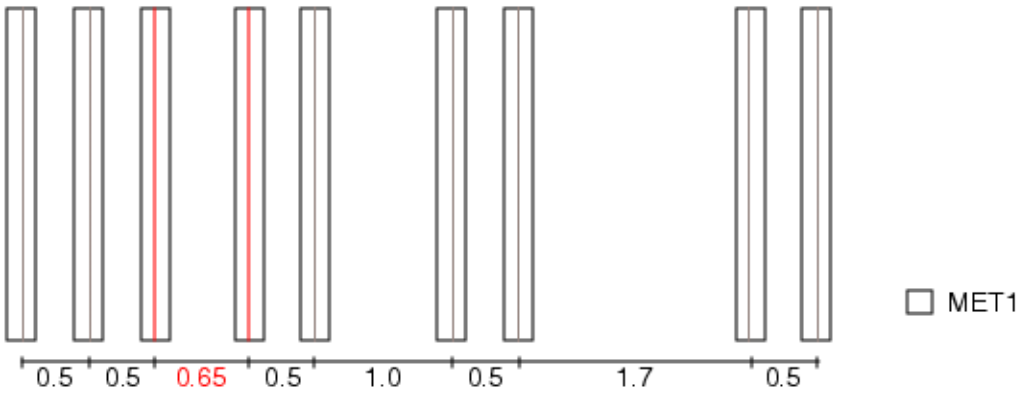
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Checking Metal Routing Relative Pitch

You can check that the centerline-to-centerline spacing of routing wires are spaced with an integer multiple of the pitch value relative to each other.

This rule does not apply to distances greater than the optical radius value of 1.5. As shown in the following figure, the wires spaced 0.65 units apart are identified as errors because they are not spaced with an integer multiple of the pitch value of 1 user unit. However, the wires spaced 1.7 units apart pass the check because they are outside the optical radius.

Figure 2-24. Relative Pitch Checking for Metal Routing



Try It!	**Calibre Advanced DRC (eqDRC) Tutorial and Example Kit**
	Go to this page on Support Center to download the complete eKit.
	This example is in the grid_pitch_checks/metal_routing example directory.

Procedure

1. Define the pitch, optical radius, and maximum metal width values.

```
VARIABLE optradius 1.5
VARIABLE pitch 0.5
VARIABLE metal_wid 0.4
```

2. Generate centerlines for each metal line.

```
cl_widths = DFM SPACE M1 <= metal_wid BY INT HORIZONTAL
cl = DFM COPY cl_widths CENTERLINE
```

3. Identify off-pitch centerlines. The ALL keyword must be used because DFM COPY with CENTERLINE generates edges that face arbitrary directions.

```
cl_space = DFM SPACE cl <= optradius ALL HORIZONTAL GRID pitch
```

4. Attach properties containing pitch calculations using a DFM Property operation. The properties generated tell you the spacing distance, by what value the spacing should be

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

changed, the deviation from the correct pitch value as a percentage, and the number of pitch values.

```
metal_pitch_err = DFM PROPERTY cl_space
     [distance = EW(cl_space)]
     [change_space_by = REMAINDER(EW(cl_space),pitch)]!= 0
     [pct_off_pitch =
         ROUND(ABS(100*(REMAINDER(EW(cl_space),pitch)/pitch)),0.1)]
     [num_pitches = ROUND(EW(cl_space)/pitch,0.1)]
```

5. Output the results using a DFM RDB operation.

```
m1_rel_pitch {
  DFM RDB metal_pitch_err "eqdrc.rdb" CELL SPACE ALL CELLS NOPSEUDO
      CHECKNAME "%_l_"
}
```

Related Topics

Introduction to DFM Property [Calibre Verification User's Manual]

Measurement Functions [Standard Verification Rule Format (SVRF) Manual]

Supported Math Functions in DFM Expressions [Standard Verification Rule Format (SVRF) Manual]

Introduction to DFM RDB [Calibre Verification User's Manual]

Math Operators [Standard Verification Rule Format (SVRF) Manual]

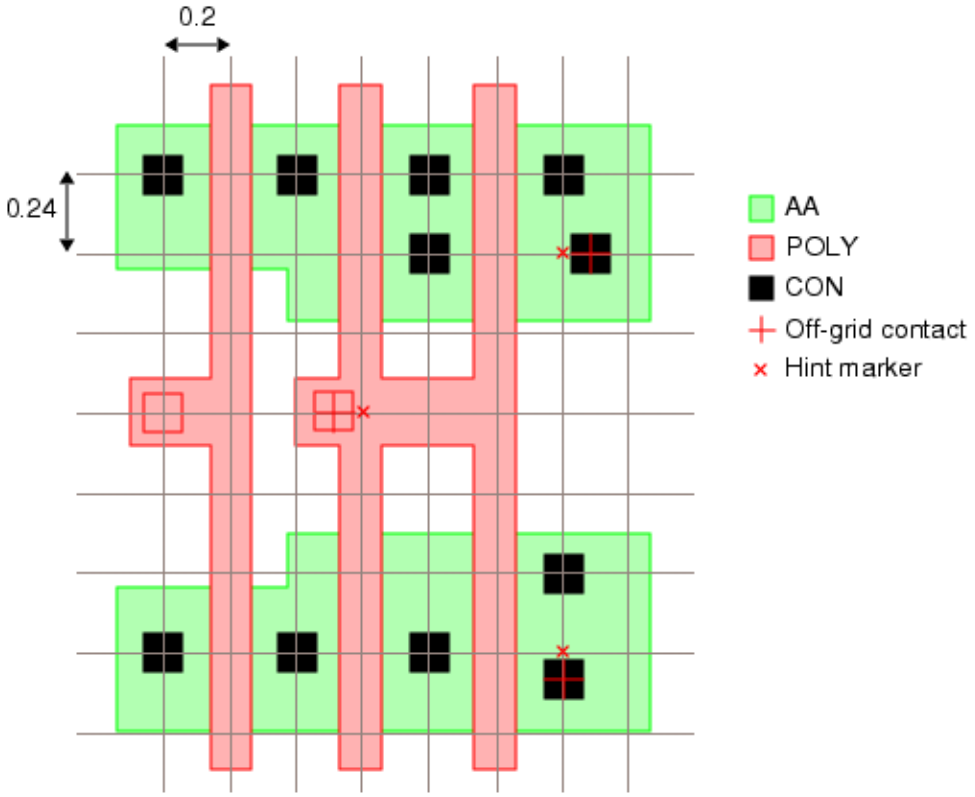
Checking Contact Relative Pitch

You can check that the center points of contacts within an optical radius are spaced with an integer pitch value relative to each other.

Refer to the conceptual illustration shown in Figure 2-25. In this figure, contacts that are not aligned to the grid (which is spaced 0.2 units in the x direction and 0.24 units in the y direction) are identified as errors, and hint markers show how each contact can be moved to pass the check.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 2-25. Contact Relative Pitch



This check can be implemented using the Offgrid operation.

Try It!	**Calibre Advanced DRC (eqDRC) Tutorial and Example Kit**
	Go to this page on Support Center to download the complete eKit.
	This example is in the grid_pitch_checks/con_rel_pitch example directory.

Procedure

1. Define the contact pitch, optical radius, and contact width.

```
VARIABLE xpitch 0.20 // contact pitch in x direction
VARIABLE ypitch 0.24 // contact pitch in y direction
VARIABLE optradius 2 // optical radius distance
VARIABLE contwidth 0.12 // contact width
```

2. Define the layer that represents the optical radius. This layer is used to group contacts separated by a distance that is less than the optical radius.

```
cogroup = EXTENTS (GROW CONT
                    RIGHT BY xpitch*optradius-contwidth/2
                    LEFT BY xpitch*optradius-contwidth/2
                    TOP BY ypitch*optradius-contwidth/2
                    BOTTOM BY ypitch*optradius-contwidth/2
                   )
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

3. Use the Offgrid command to perform grid checking for the groups of contacts.

```
offgridcont = OFFGRID CONT 200 240 INSIDE OF LAYER cogroup ABSOLUTE
              CENTERS HINT
```

4. Output the results with a DFM RDB operation.

```
con_pitch {
  DFM RDB offgridcont "contactpitch.rdb" CHECKNAME "%_l_"
      CELL SPACE ALL CELLS NOPSEUDO
}
```

Related Topics

Introduction to DFM Property [Calibre Verification User's Manual]

Measurement Functions [Standard Verification Rule Format (SVRF) Manual]

Supported Math Functions in DFM Expressions [Standard Verification Rule Format (SVRF) Manual]

Introduction to DFM RDB [Calibre Verification User's Manual]

Math Operators [Standard Verification Rule Format (SVRF) Manual]

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Miscellaneous Advanced DRC Checks

This section includes miscellaneous advanced DRC checks.

Table 2-5. Miscellaneous DRC Check Solutions

Name	Description	SVRF Elements
Checking Dangling Wires	Identifies sections of metal that connect to one or more vias at one end, and are unconnected at the other end.	External, DFM Space, DFM Copy, XOR, Convex Edge, Internal, NOT, DFM Property, Expand Edge, Interact, DFM RDB
Finding Circular Polygons	Finds polygons that are approximately circular using two different methods.	Inside Cell, DFM Circle Analyze, DFM Property, DFM RDB
Finding Length and Width Edges of Rectangles	Identifies length and width edges of rectangular polygons that are not square.	Length, Rectangle, DFM Property
Finding Pad Edges Closest to the Chip Edge	Identifies edges on pads that are closest to the edge of the chip.	Enclosure, Expand Edge, DFM Property, DFM RDB
Finding Row and Column Counts in an Array	Finds the number of rows and columns in an array of shapes.	Convex Edge, DFM Copy, DFM Property, DFM RDB, Grow, Extents, External, NOT
Finding Polygon Extents for Special Configurations	Determines polygon extents for cases where extents overlap. An unmerged polygon layer containing the polygon extents is also generated.	DFM Property, DFM Property Select Secondary, DFM RDB
Finding Unique Violations	Finds unique violations from a set of DRC violations. The rule check outputs only one error marker from a set of redundant violations. The redundant violations occur when more than one spacing violation occurs between the same two polygons. This is useful for reducing the number of DRC results.	External, Internal, DFM Classify, DFM Property, DFM Space, DFM RDB
Resizing Polygons by a Percentage of the Area	Resizes polygons by a percentage of their area.	DFM Property, DFM Size, DFM RDB

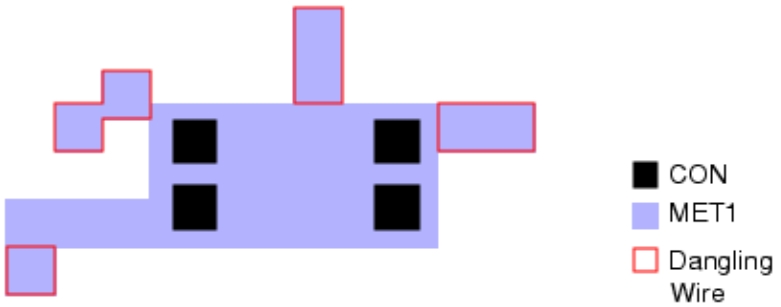
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Checking Dangling Wires

A dangling wire is defined as a segment of metal that connects to one or more vias at one end and is unconnected at the other end.

Figure 2-26 shows dangling wires for an example MET1 polygon. You can check for dangling wires in your design by first breaking a metal layer into segments (separated by small gaps), then determining which portions of the metal layer are dangling based on interaction between the segments and the metal.

Figure 2-26. Dangling Wires



Try It!	**Calibre Advanced DRC (eqDRC) Tutorial and Example Kit**
	Go to this page on Support Center to download the complete eKit.
	This example is in the miscellaneous_checks/dangling_wire example directory.

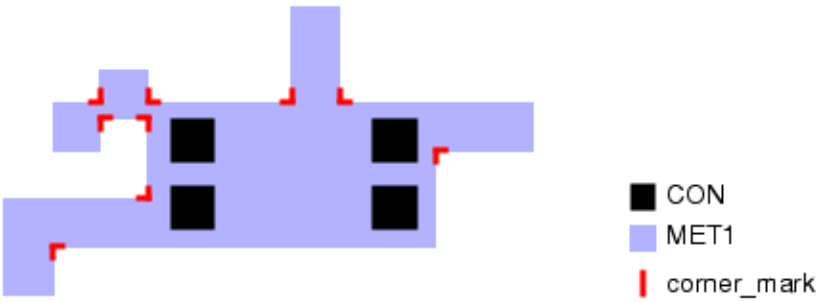
Procedure

1. Create a variable for the widths of the derived gaps and the minimum metal width.

```
VARIABLE dist 0.002 // gap width
VARIABLE min_w 0.05 // minimum wire width
```

2. Derive single edges of length dist on each of the inside corners.

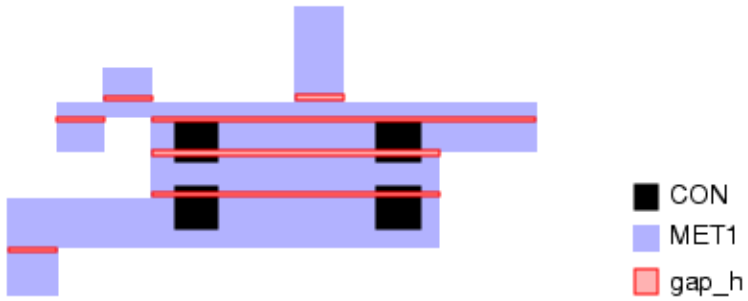
```
corner_mark = EXT [MET1] < dist ABUT == 90
```



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

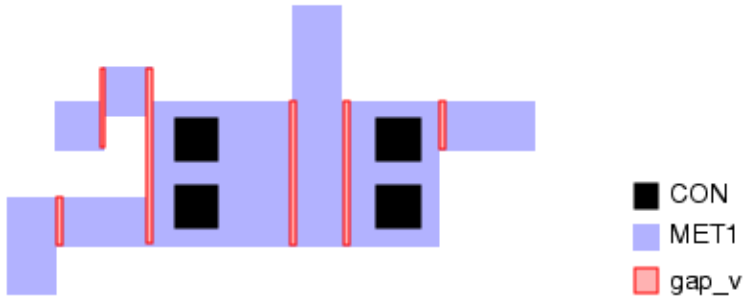
3. Use the corner edges to derive small horizontal gaps that separate the MET1 shape into rectangles.

```
concave_pair_h = DFM SPACE corner_mark MET1 < 3 BY INT HORIZONTAL
                 MEASURE ALL
gap_h = DFM COPY concave_pair_h REGION
```



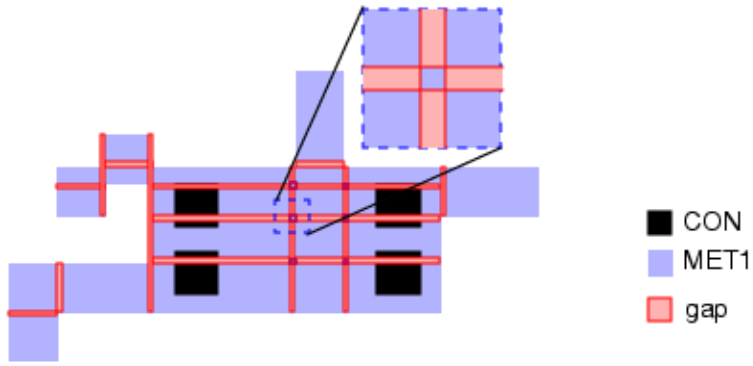
4. Repeat the previous step for the vertical direction.

```
concave_pair_v = DFM SPACE corner_mark MET1 < 3 BY INT VERTICAL
                 MEASURE ALL
gap_v = DFM COPY concave_pair_v REGION
```



5. Combine the horizontal and vertical gaps and filter out the intersection to prevent them from merging.

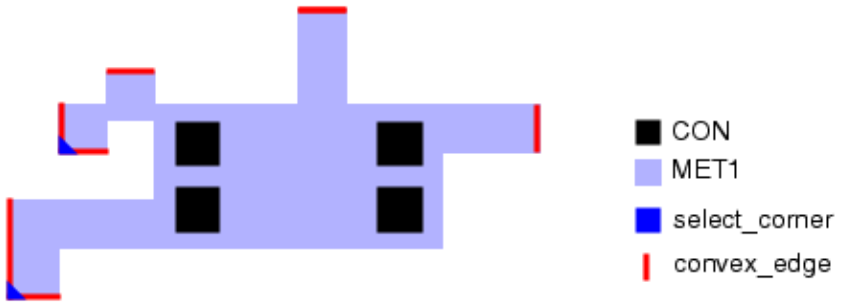
```
gap = gap_v XOR gap_h
```



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

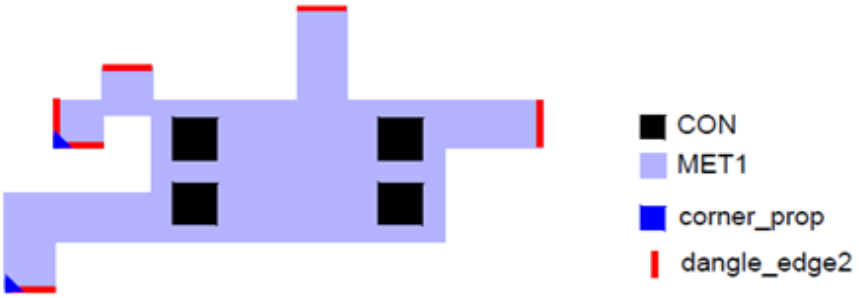
6. Derive the outside edges (edges with two convex end points). Put corner markers where two outside edges meet.

```
convex_edge = CONVEX EDGE MET1 == 2
select_corner = INT convex_edge < dist ABUT == 90 REGION
```



7. In cases where two convex_edges meet, select the shorter edge to represent the dangling tip. It is possible for a convex_edge to touch 0, 1, or 2 corner markers.

```
// for two convex edges that meet, determine the shorter edge and
// attach the length as a property to the corner marker shape
corner_prop = DFM PROPERTY select_corner convex_edge
                  OVERLAP ABUT ALSO MULTI
                  [short = MIN(LENGTH(convex_edge))]

// filter for shorter edges
dangle_edge2 = DFM PROPERTY convex_edge corner_prop
                   OVERLAP ABUT ALSO MULTI
// output all convex edges that do not form a corner
// with another convex edge
[ "+test0" = (COUNT(corner_prop) == 0) ? 1 : 0 ]
// for edges that touch one corner_prop shape, output the
// shorter edge
[ "+test1" = (COUNT(corner_prop) != 1) ? 0 :
   (LENGTH(convex_edge) == PROPERTY(corner_prop,short, 1)) ? 1 : 0]
// for edges that touch two corner_prop shapes, output the
// shorter edge
[ "+test2" = (COUNT(corner_prop) != 2) ? 0 :
  (LENGTH(convex_edge) == PROPERTY(corner_prop,short, 1)) ? 1 :
   (LENGTH(convex_edge) == PROPERTY(corner_prop,short, 2)) ? 1 : 0 ]
// output edges that meet any of the conditions
[ - = PROPERTY_REF("+test0") + PROPERTY_REF("+test1") +
              PROPERTY_REF("+test2") ] > 0
```



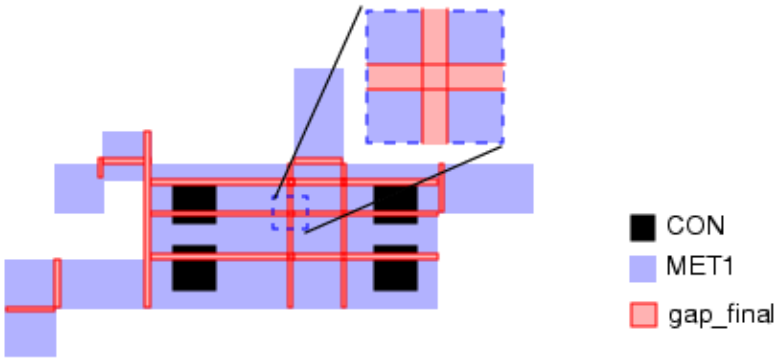
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

8. Expand the dangling tip edges into polygons. Filter the gaps that touch a dangling tip.

```
dangle_tip = EXPAND EDGE dangle_edge2 INSIDE BY dist
gap_notip = gap NOT INTERACT dangle_tip
```

9. Derive the intersection pieces, then combine them with the gap polygons to create continuous shapes.

```
gap_ext = EXPAND EDGE (LENGTH gap_notip == dist) OUTSIDE BY dist
gap_final = DFM COPY gap_notip gap_ext
```

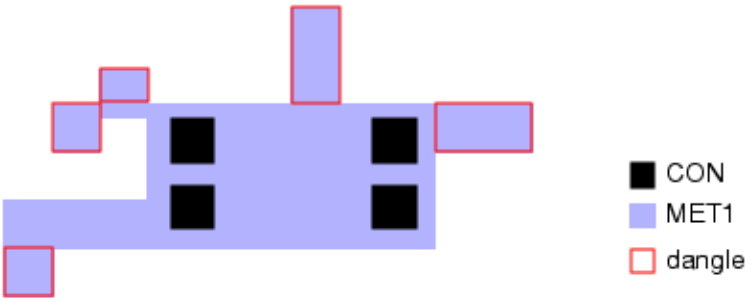


10. Derive the MET1 sections that are separated by each gap.

```
MET1_pieces = MET1 NOT gap_final
```

11. Find each dangling polygon by identifying the metal sections that interact with the dangling tips.

```
dangle = MET1_pieces INTERACT dangle_tip
```



12. Filter out dangling polygons that contain contacts.

    Then, size up the result by the gap width dist. This prevents the not_dangle layer that is derived in the next step from having slivers of width dist.

```
dangle2 = dangle NOT INTERACT v1
dangle_all = SIZE dangle2 BY dist INSIDE OF m2
```

13. Derive non-dangling polygons, and filter out special cases where a dangling wire connects to a non-dangling section in more than one place.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

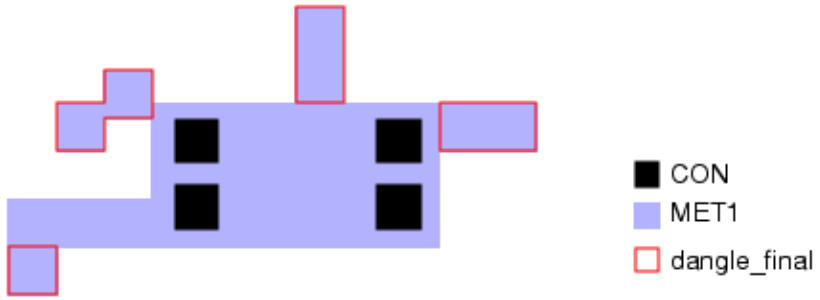
```
not_dangle = MET1 NOT dangle_all
dangle_wire2 = (INTERACT dangle_all not_dangle == 1) OR
               (NOT INTERACT dangle_all not_dangle)
```

14. In certain cases, removing a dangling section creates sections of metal that are too narrow. In these cases, use the min_w constraint to include these sections in the output.

```
not_dangle2 = MET1 NOT dangle_wire2
min_width = INT not_dangle2 < dist OPPOSITE
min_width_region = DFM COPY min_width REGION
not_min_width_region = not_dangle2 NOT min_width_region

jog_filt = INTERACT min_width_region not_min_width_region == 2
dangle_wire3 = NOT INTERACT dangle_wire2 jog_filt
```

15. Derive the final result.

```
dangle_final = DFM COPY dangle_wire3 (min_width_region NOT jog_filt)
```



16. Output the results with a DFM RDB operation.

```
dangling_wires {
    DFM RDB dangle_final advDRC.rdb CHECKNAME "%_l_"
        CELL SPACE ALL CELLS
}
```

Related Topics

Introduction to DFM Property [Calibre Verification User's Manual]

Introduction to DFM Function [Calibre Verification User's Manual]

Measurement Functions [Standard Verification Rule Format (SVRF) Manual]

Supported Math Functions in DFM Expressions [Standard Verification Rule Format (SVRF) Manual]

Introduction to DFM RDB [Calibre Verification User's Manual]

Conditional Expressions [Standard Verification Rule Format (SVRF) Manual]

Math Operators [Standard Verification Rule Format (SVRF) Manual]

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Finding Circular Polygons

You can use a DFM Circle Analyze or Inside Cell CIRCLE operation to find approximately circular shapes in a layout.

- DFM Circle Analyze — For each polygon on the input layer, the operation finds the minimum enclosing circle C-out and the largest circle C-in that is contained within the polygon and concentric with C-out. The radii of these two circles are reported as DFM properties named OUTRADIUS and INRADIUS attached to each shape on the output layer.

  The OUTRADIUS and INRADIUS properties can be used in a DFM Property operation to select shapes that meet a desired criteria.

  DFM Circle Analyze merges the input layer before shapes are analyzed.

- Inside Cell CIRCLE — The CIRCLE keyword causes the Inside Cell operation to return shapes that meet a built-in criteria. In particular, the angle between all polygon vertices must vary by less than 1%, and the distance between the center of the polygon extent and every vertex must vary by less than 1%. See the command definition for additional criteria.

  Inside Cell operates on unmerged original layer input, so it can output drawn circular shapes that overlap other shapes on the same layer. However, the output is merged, so overlapping shapes are merged on output.

The circle criteria for Inside Cell is built-in and cannot be changed, and is also very strict. When using DFM Circle Analyze, however, you can use the OUTRADIUS and INRADIUS properties to develop your own criteria for a circular shape.

Try It!	**Calibre Advanced DRC (eqDRC) Tutorial and Example Kit**
	Go to this page on Support Center to download the complete eKit.
	This example is in the miscellaneous_checks/finding_circles example directory.

Procedure

1. Use DFM Circle Analyze to attach the OUTRADIUS and INRADIUS properties to the output polygon layer. Use a DFM Property operation to select polygons with a ratio INRADIUS/OUTRADIUS > 0.9.

```
circle_props = DFM CIRCLE ANALYZE MET1
circles_ratio_90 = DFM PROPERTY circle_props
              [ - = PROPERTY(circle_props, INRADIUS)/
                    PROPERTY(circle_props, OUTRADIUS)] > 0.90
         // for 2020.3 and later, prevent selecting annular shapes
              [ - = PROPERTY(circle_props, ANNULUS_INRADIUS)] == 0
```

2. Use Inside Cell to select original layer polygons that meet the built-in CIRCLE criteria.

```
circles_inside_cell = INSIDE CELL MET1 "*" CIRCLE
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

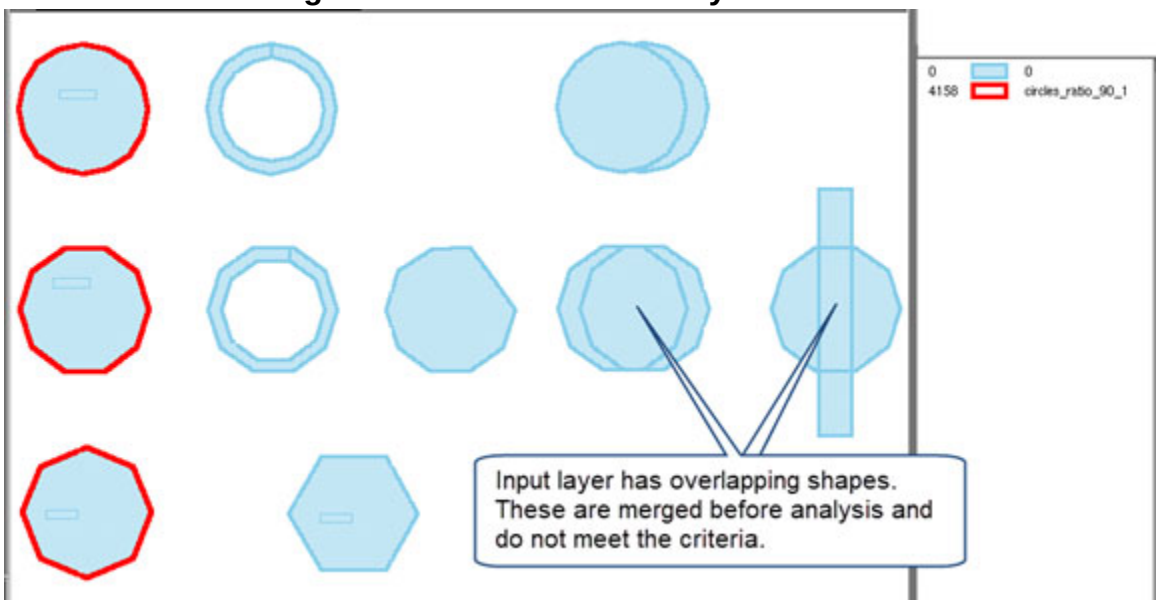
3. Output the results with a DFM RDB operation.

```
circles {
    DFM RDB circles_ratio_90 circles.rdb CHECKNAME "%_l_"
        CELL SPACE ALL CELLS
    DFM RDB circles_inside_cell circles.rdb CHECKNAME "%_l_"
        CELL SPACE ALL CELLS
}
```

Results

These are the results from the DFM Circle Analyze operation. If the filter criteria is changed to INRADIUS/OUTRADIUS > 0.95, only the top left shape is returned.

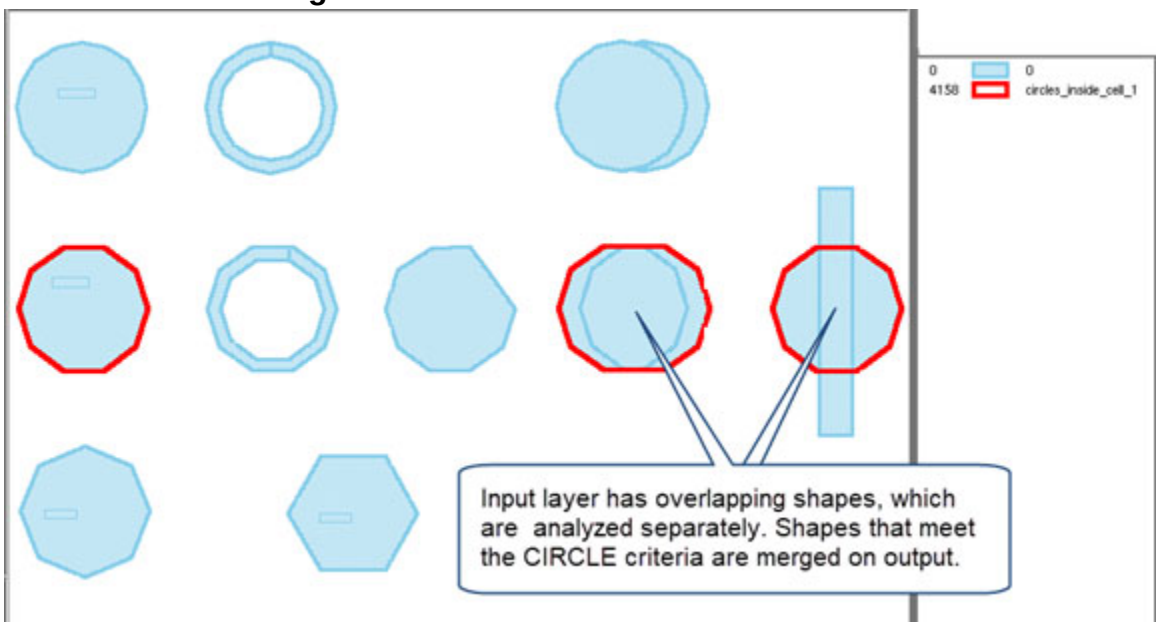
Figure 2-27. DFM Circle Analyze Results



These are the results from the Inside Cell CIRCLE operation. Due to the strict criteria of the CIRCLE keyword, the Inside Cell operation does not output the shape at the top left, even though it appears circular to the eye.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 2-28. Inside Cell CIRCLE Results



Related Topics

Introduction to DFM Property [Calibre Verification User's Manual]

Measurement Functions [Standard Verification Rule Format (SVRF) Manual]

Introduction to DFM RDB [Calibre Verification User's Manual]

Math Operators [Standard Verification Rule Format (SVRF) Manual]

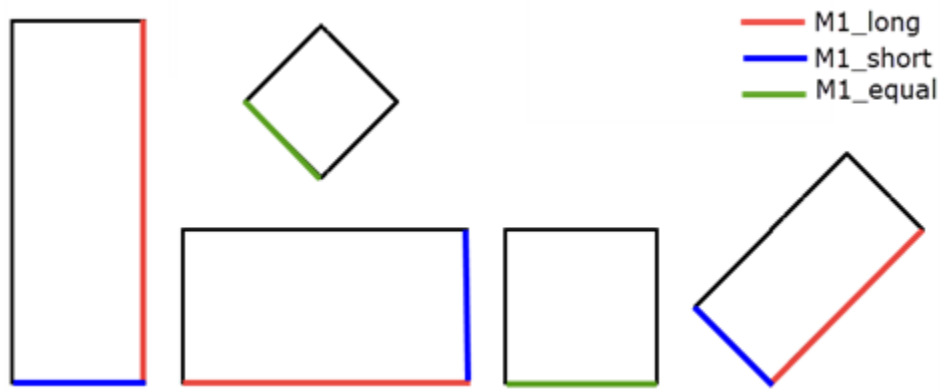
Finding Length and Width Edges of Rectangles

You can find length and width edges of any rectangle. Length edges are defined as the longer edges and width edges are the shorter edges. You can also identify squares and output one of the edges.

The following figure shows a conceptual illustration. For this check, one edge of each type (long, short, or equal) is output per polygon. The check can easily be changed to output all edges of a specific type.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 2-29. Length and Width Edges of Rectangles



The check uses a single-layer DFM Property Select Secondary operation. The primary layer is the polygon layer of interest. The :EDGE layer derivation is used to output edges of the primary layer shapes that match the specified conditions.

Try It!	**Calibre Advanced DRC (eqDRC) Tutorial and Example Kit**
	Go to this page on Support Center to download the complete eKit. This example is in the miscellaneous_checks/rectangle_edges example directory.

Procedure

1. Filter out metal1 shapes that are not rectangles.

```
M1 = RECTANGLE metal1
```

   If you want to consider all metal1 polygons, skip this step.

2. Find and output a long edge of the polygon.

```
M1_long = DFM PROPERTY SELECT SECONDARY M1
                SELECT M1:EDGE NOMULTI
                [ - = ( DRC_NE(MIN(LENGTH(M1:EDGE)),
                              MAX(LENGTH(M1:EDGE))) > 0) ? 1 : 0 ]
                            == 1
                [ - = (LENGTH(M1:EDGE) == MAX(LENGTH(M1:EDGE)))
                            ? 1 : 0 ] == 1
                [long_len = LENGTH(M1:EDGE)]
```

   “SELECT M1:EDGE NOMULTI” instructs the operation to output one edge of the M1 layer shape, where the edge must meet the subsequent property conditions. Remove NOMULTI if you want to output all edges that meet the conditions.

   The first conditional expression limits the output to polygons with unequal edge lengths DRC_NE() returns 1.0 if its arguments are not equal within a built-in tolerance, and returns 0.0 otherwise.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The second conditional expression limits the output to edges with a length equal to the maximum edge length.

The edge length is attached as the property long_len.

3. Find and output a short edge of the polygon.

```
M1_short = DFM PROPERTY SELECT SECONDARY M1
                SELECT M1:EDGE NOMULTI
                [ - = ( DRC_NE(MIN(LENGTH(M1:EDGE)),
                               MAX(LENGTH(M1:EDGE))) > 0) ? 1 : 0 ]
                             == 1
                [ - = (LENGTH(M1:EDGE) == MIN(LENGTH(M1:EDGE)))
                             ? 1 : 0 ] == 1
                [short_len = LENGTH(M1:EDGE)]
```

4. Find polygons that have edges with equal lengths, and output one edge.

```
M1_equal = DFM PROPERTY SELECT SECONDARY M1
                        SELECT M1:EDGE NOMULTI
                [ - = ( DRC_EQ(MIN(LENGTH(M1:EDGE)),
                               MAX(LENGTH(M1:EDGE))) > 0 ) ? 1 : 0 ]
                            == 1
                [equal_len = LENGTH(M1:EDGE)]
```

The conditional expression with DRC_EQ() limits the output to polygons that have equal edge lengths.

It is not necessary to select edges of a specific length, since they are all equal. As with the preceding derivations, remove NOMULTI to output all edges that meet the conditions.

Suppose the M1 layer contains only polygons that are orthogonal to the database axes, and you want output the horizontal edge. You can add this condition:

```
// for orthogonal input, include this to output the horizontal edge
[ - = (LENGTH(M1:EDGE) == LENGTHX(M1:EDGE)) ? 1 : 0 ] == 1
```

5. Output the results with a DFM RDB operation.

```
// Set default options for DFM RDB operations
DFM DEFAULTS RDB CELL SPACE ALL CELLS CHECKNAME "%_l_"
// output the results
rectangle_edges {
    DFM RDB M1_long eqdrc.rdb
        COMMENT "long edge of rectangles on metal1"
    DFM RDB M1_equal eqdrc.rdb
        COMMENT "equal length edge of rectangles on metal1"
    DFM RDB M1_short eqdrc.rdb
        COMMENT "short edge of rectangles on metal1"
}
```

The derived edge layers can be used in further layer operations.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

DRC Constraint Comparison Functions [Standard Verification Rule Format (SVRF) Manual]

Conditional Expressions [Standard Verification Rule Format (SVRF) Manual]

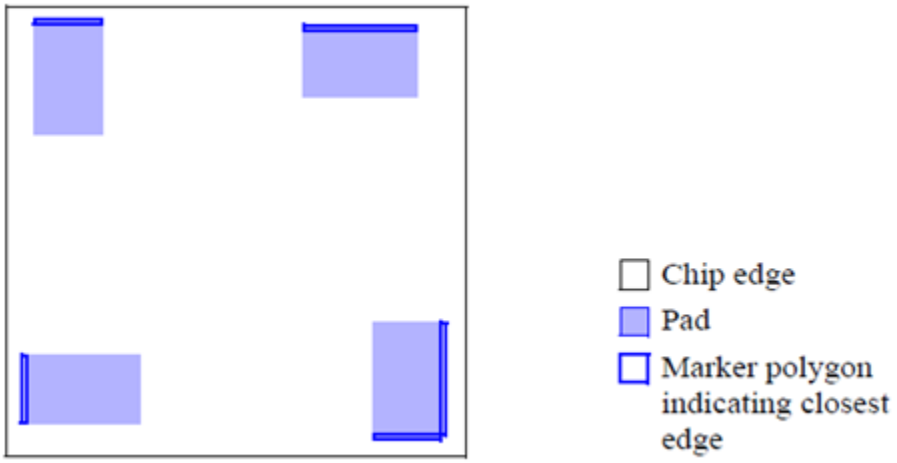
Introduction to DFM Property [Calibre Verification User's Manual]

Measurement Functions [Standard Verification Rule Format (SVRF) Manual]

Finding Pad Edges Closest to the Chip Edge

You can identify edges of pads that are closest to the chip edge using marker polygons. This check can be useful for determining which way pads should be oriented.

Figure 2-30. Pad Edges Closest to the Chip Edge



Try It!	**Calibre Advanced DRC (eqDRC) Tutorial and Example Kit**
	Go to this page on Support Center to download the complete eKit. This example is in the miscellaneous_checks/pads_chip_edge example directory.

Procedure

1. Using the Enclosure operation, derive edge pairs that measure the separation from the exterior-facing sides of the pad layer to the interior-facing sides of the chipedge layer.

```
VARIABLE enc_dist 0.15
enc_error = ENC pad chipedge <= enc_dist OPPOSITE
```

Use an appropriate measurement constraint for your design.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

2. Expand by a small amount each pad edge that faces the chip edge to create marker polygons:

```
enc_region = EXPAND EDGE (ENC [pad] chipedge <= enc_dist OPPOSITE)
               OUTSIDE BY 0.001
```

   The use of brackets ([]) surrounding the pad layer in the Enclosure operation causes edge-directed output, rather than the default error-directed output.

3. Attach DFM properties to the marker polygons to indicate the distance between each marker polygon and the chip edge.

```
enc_value = DFM PROPERTY enc_region enc_error OVERLAP ABUT ALSO
              [ChipEdgeDistance = MIN(EW(enc_error))]
```

4. Identify the pad edges closest to the chip edge. To do this, use the MIN() function to determine the smaller distances, then transfer the DFM properties to the pad layer.

```
enc_min = DFM PROPERTY pad enc_value OVERLAP ABUT ALSO
              [ChipEdgeDistanceMin =
                 MIN(PROPERTY(enc_value,ChipEdgeDistance))]
```

5. Identify the marker polygons that correspond to the closest edges. To do this, first, for each marker polygon, divide each distance by the minimum distance. Then, use a constraint of “==1” to select the marker polygons with the minimum distance. The non-persistent property name “-” is used because the value is not of interest.

   Output the chip to pad distance as the property ChipEdgeDistance.

```
closer_edge = DFM PROPERTY enc_value enc_min OVERLAP ABUT ALSO MULTI
                 [- = PROPERTY(enc_value,ChipEdgeDistance)/
                      PROPERTY(enc_min,ChipEdgeDistanceMin)] == 1
                 [ChipEdgeDistance =
                      PROPERTY(enc_value,ChipEdgeDistance)]
```

   If a pad at the chip corner has edges with equal distances to the chip edge, both marker polygons are output.

6. Output the results of the DFM Property operation using DFM RDB.

```
pad_edge {
    DFM RDB closer_edge advDRC.rdb CHECKNAME "%_l_"
       CELL SPACE ALL CELLS
}
```

Related Topics

Introduction to DFM Property [Calibre Verification User's Manual]

Measurement Functions [Standard Verification Rule Format (SVRF) Manual]

Introduction to DFM RDB [Calibre Verification User's Manual]

Math Operators [Standard Verification Rule Format (SVRF) Manual]

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Finding Row and Column Counts in an Array

You can find the number of rows and columns in an array of shapes using a DFM Property operation. The values are attached to the output layer as DFM properties. The number of shapes in each row and column is also determined.

The shapes in the array are assumed to be Manhattan shapes. If they are not, you can use the following derivation on the extent of each shape in the array. This derivation does not check for uniform shape size or spacing.

Try It!	**Calibre Advanced DRC (eqDRC) Tutorial and Example Kit**
	Go to this page on Support Center to download the complete eKit.
	This example is in the miscellaneous_checks/array_counts example directory.

Procedure

1. Define the following variables for the maximum length of a shape in an array and the maximum space between two shapes in the same array:

```
VARIABLE max_shape_len 0.6     // max edge length of a shape
VARIABLE max_shape_space 0.7   // max space between two shapes
                               // in the same group
```

2. Derive a layer containing all shape edges that are shorter than max_shape_len.

```
LAYER shape 0
shape_edges = LENGTH shape < max_shape_len
```

Note

The shape layer can be used instead of shape_edges in the following derivation if it is not necessary to filter shapes by edge length.

3. Construct a macro that returns the spaces between edges or polygons in a row or column. The macro uses DFM Space with BY EXT and HORIZONTAL or VERTICAL to get edge pairs in a row or column. DFM Copy with the REGION keyword generates polygons from the edge pairs.

```
// Macro to return spaces in a row or column
// "direction" is HORIZONTAL for spaces in a row,
//             or VERTICAL for a column
// "dist" is the max distance for shapes in the same row or column
DMACRO SPACES "in" "dist" "direction"  "out" {
         edge_pairs = DFM SPACE "in" < "dist" BY EXT "direction"
         "out" = DFM COPY edge_pairs REGION
}
```

4. Find rows and columns of shapes and attach the shape count to the output layer. Shapes within the max_shape_space distance form a row or column.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Use the SPACES macro to get the spaces between shapes in a row. Use DFM Copy to combine the shapes and spaces into a row. Use DFM Property to count the number of shapes in each row.

```
CMACRO SPACES shape_edges max_shape_space HORIZONTAL row_spaces
rows = DFM COPY shape row_spaces

rows_prop = DFM PROPERTY rows shape
                [shape_count = COUNT(shape)]
```

Do the same for columns.

```
CMACRO SPACES shape_edges max_shape_space VERTICAL col_spaces
col = DFM COPY shape col_spaces

col_prop = DFM PROPERTY col shape
               [shape_count = COUNT(shape)]
```

5. Group together all rows and columns that are within the max_shape_space distance of each other.

```
 // get spaces between each row
CMACRO SPACES rows max_shape_space VERTICAL row_vert_spaces

// get spaces between each column
CMACRO SPACES col max_shape_space HORIZONTAL col_horiz_spaces

// combine rows, columns, and the spaces between them
group_all = DFM COPY rows col row_vert_spaces col_horiz_spaces
```

The group_all layer overlaps each array of shapes.

6. Use DFM Property to find the row and column count for each array of shapes. The minimum and maximum number of shapes in each row and column is also calculated, along with the total number of shapes in the array.

```
row_col_count = DFM PROPERTY group_all rows_prop col_prop OVERLAP
       [row_count = COUNT(rows_prop)]
       [column_count = COUNT(col_prop)]
       [min_shapes_in_row = MIN(PROPERTY(rows_prop, shape_count))]
       [max_shapes_in_row = MAX(PROPERTY(rows_prop, shape_count))]
       [min_shapes_in_col = MIN(PROPERTY(col_prop, shape_count))]
       [max_shapes_in_col = MAX(PROPERTY(col_prop, shape_count))]
       [total_shapes = SUM(PROPERTY(col_prop, shape_count))]
```

Although not done in this example, any of the property values can be used to create an filter condition.

7. Output the results to an RDB for viewing.

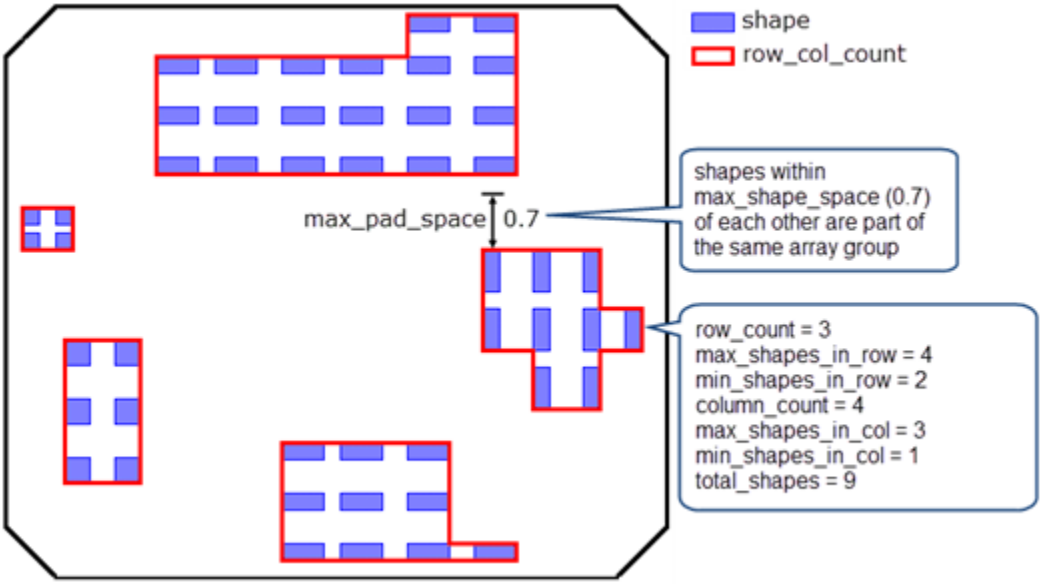
```
row_col_count {
    DFM RDB row_col_count eqdrc.rdb CHECKNAME "%_l_"
      CELL SPACE ALL CELLS
      COMMENT "Counts of rows and columns in each group,  "
      COMMENT "with min, max, and total counts."
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Results

The following image demonstrates the results for a layout with several different arrays.

Figure 2-31. Finding Row and Column Counts



Related Topics

Finding Pad Edges Closest to the Chip Edge

Introduction to DFM Property [Calibre Verification User's Manual]

Measurement Functions [Standard Verification Rule Format (SVRF) Manual]

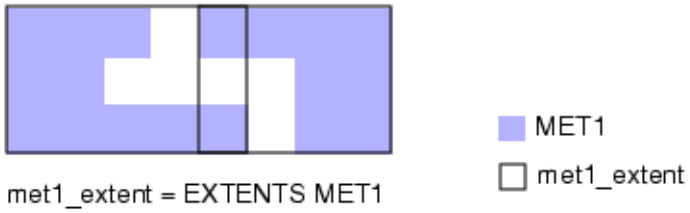
Introduction to DFM RDB [Calibre Verification User's Manual]

Finding Polygon Extents for Special Configurations

In certain cases, polygons on a layout can be arranged in such a way that finding their extents cannot be done using a single Extents operation.

In the following figure, the extents of the two MET1 polygons overlap, but the polygons themselves do not. Since the output of the Extents operation is merged, the extents of the individual polygons cannot be determined.

Figure 2-32. Overlapping Polygon Extents



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The DFM Property operation supports the application of the :EXTENT_EDGE layer derivation to the input layer of measurement functions. This can be used to calculate the x- and y- extent of each polygon on the input layer and attach the values as DFM properties on each polygon.

In addition, the DFM Property Select Secondary operation can use the :EXTENT layer derivation to output an unmerged polygon layer that contains the extent bounding box of each shape on the input layer. Unmerged polygons layers are supported by a limited number of operations.

Try It!	**Calibre Advanced DRC (eqDRC) Tutorial and Example Kit**
	Go to this page on Support Center to download the complete eKit. This example is in the miscellaneous_checks/polygon_extents example directory.

The following procedure uses both approaches (DFM Property and DFM Property Select Secondary).

Procedure

1. Use DFM Property to calculate the polygon extent in the x- and y-directions. The output layer is a copy of the input layer with DFM properties attached.

   The :EXTENT_EDGE layer derivation returns the edges of the bounding box of the object on the input layer. The edges are returned as an error layer. The ECX and ECY measurement functions calculate the total x and y lengths of the input edges.

   ```
   MET1_extent_prop = DFM PROPERTY MET1
                          [horizlength = ECX(MET1:EXTENT_EDGE)/2]
                          [vertlength  = ECY(MET1:EXTENT_EDGE)/2]
   ```

2. Use DFM Property Select Secondary to output the extent bounding boxes as an unmerged polygon layer. The SELECT keyword specifies the MET1 layer with the :EXTENT layer derivation. For this example the horizontal and vertical lengths are calculated using the :EXTENT layer derivation and the PERIMX and PERIMY measurement functions.

   ```
   MET1_bbox_unmerged = DFM PROPERTY SELECT SECONDARY MET1
                            [horizlength = PERIMX(MET1:EXTENT)/2]
                            [vertlength  = PERIMY(MET1:EXTENT)/2]
                            SELECT MET1:EXTENT
   ```

3. Output the results with a DFM RDB operation.

   ```
   met1_extent {
         DFM RDB MET1_extent_prop advDRC.rdb CHECKNAME "%_l_"
             CELL SPACE ALL CELLS
         DFM RDB MET1_bbox_unmerged advDRC.rdb CHECKNAME "%_l_"
             CELL SPACE ALL CELLS
   }
   ```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

:EDGE, :EXTENT, and :EXTENT_EDGE Layer Derivation Indicators [Standard Verification Rule Format (SVRF) Manual]

ECX [Standard Verification Rule Format (SVRF) Manual]

Introduction to DFM Property [Calibre Verification User's Manual]

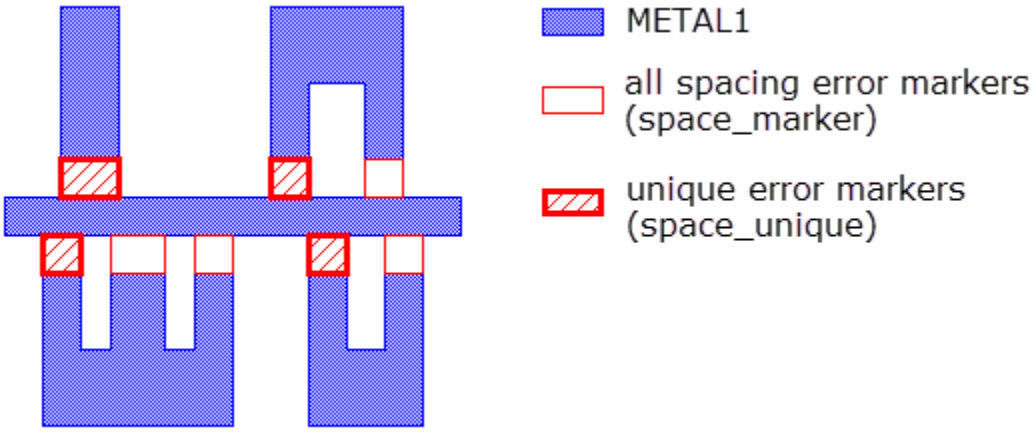
Introduction to DFM RDB [Calibre Verification User's Manual]

Unmerged Polygon Layers [Calibre Verification User's Manual]

Finding Unique Violations

You can use DFM Property and DFM Classify to output only unique violations from a set of DRC violations. The example constructs error marker polygons using an External operation. The rule check outputs only one error marker from a set of redundant violations. The redundant violations occur when more than one spacing violation occurs between the same two polygons. This is useful for reducing the number of DRC results.

Figure 2-33. Example of Unique Spacing Violations Between Polygons



Try It!	**Calibre Advanced DRC (eqDRC) Tutorial and Example Kit**
	Go to this page on Support Center to download the complete eKit.
	This example is in the miscellaneous_checks/unique_violations example directory.

Procedure

1. Derive spacing error markers between the polygons on METAL1.

```
space_marker = EXT METAL1 <= 0.1 ABUT REGION
```

2. Generate a unique property set on each METAL1 polygon.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The properties proph and propv contain unique coordinate values for each polygon.

```
corners = INT METAL1 < 0.05 ABUT REGION
epairsh = DFM SPACE corners <= 0.05 BY INT HORIZONTAL
epairsv = DFM SPACE corners <= 0.05 BY INT VERTICAL

METAL1_props = DFM PROPERTY METAL1 epairsh epairsv OVERLAP ABUT ALSO
               [proph = MIN(ECMIN(epairsh))]
               [propv = MIN(ECMIN(epairsv))]
```

3. Copy the property sets on each METAL1 polygon to the spacing error marker that touches it, assuming that each error marker only touches two METAL1 polygons.

```
space_props = DFM PROPERTY space_marker METAL1_props
                 OVERLAP ABUT ALSO MULTI
                 [proph1 = PROPERTY(METAL1_props,proph,1)]
                 [proph2 = PROPERTY(METAL1_props,proph,2)]
                 [propv1 = PROPERTY(METAL1_props,propv,1)]
                 [propv2 = PROPERTY(METAL1_props,propv,2)]
```

4. Use DFM Classify to select one error marker per group of error markers with the same set of property values.

```
space_unique = DFM CLASSIFY space_props
                PROPERTY "proph1" "proph2" "propv1" "propv2"
                ABSOLUTE 0.001 UNIQUE KEEPPROPS
```

UNIQUE KEEPPROPS copies properties to the output layer, which may be useful for debugging.

5. Output the results with a DFM RDB operation and also write results to the standard DRC results database with a Copy operation.

```
unique_violations {
    COPY space_unique

// Set default options for DFM RDB operations
   DFM DEFAULTS RDB CELL SPACE ALL CELLS CHECKNAME "%_l_"
   DFM RDB space_unique eqdrc.rdb
   // for debug
   DFM RDB METAL1_props eqdrc.rdb
   DFM RDB space_props eqdrc.rdb
}
```

Related Topics

Introduction to DFM Property [Calibre Verification User's Manual]

Introduction to DFM RDB [Calibre Verification User's Manual]

Resizing Polygons by a Percentage of the Area

You can use eqDRC and the DFM Size operation to resize polygons by a percentage of their area.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The DFM Size operation oversizes or undersizes polygons by an amount that can be specified as a DFM property for each polygon. This procedure assumes undersizing. For a rectangular polygon with width W and length L, the shrink value must satisfy this equation:

(L - *shrink_val*)(W - *shrink_val*) = target_area

This equation can be solved using the quadratic formula to find the *shrink_val* for each polygon. The preceding equation assumes a rectangular polygon, but the equation can be generalized for use with any Manhattan polygon by using an effective width and length; see "Finding the Length of a Net Path Connected to Gate: Effective Length Method" on page 48.

The solution creates several functions with DFM Function: functions to find the effective width and length of a polygon and to solve the preceding equation for *shrink_val*. The resizing value is attached to each polygon as a DFM property. The polygons are then resized with DFM Size.

Note

The resizing method works best for rectangular polygons or polygons with a somewhat uniform width. It should only be used for Manhattan polygons.

Try It!	**Calibre Advanced DRC (eqDRC) Tutorial and Example Kit**
	Go to this page on Support Center to download the complete eKit.
	This example is in the miscellaneous_checks/size_by_percent example directory.

Procedure

1. Use DFM Function to write functions that return the effective width and length of a polygon.

```
// Find the effective length of a polygon
DFM FUNCTION [ L_EFF( LAYER L )
               (PERIM(L)+SQRT(FMAX(PERIM(L)*PERIM(L)-16*AREA(L),0)))/4
             ]
// Find the effective width of a polygon
DFM FUNCTION [ W_EFF( LAYER L )
               (PERIM(L)-SQRT(FMAX(PERIM(L)*PERIM(L)-16*AREA(L),0)))/4
             ]
```

2. Use DFM Function to write a function that solves the quadratic equation for *shrink_val*. The function is implemented with a Tcl procedure. The input arguments are the polygon effective width, effective length, and the target area.

```
// Solve this equation for shrink_val, using quadratic formula
// (L - shrink_val)(W - shrink_val) = TA
DFM FUNCTION [ QUAD_ROOT( NUMBER W, NUMBER L, NUMBER TA )
               TVF_NUM_FUN( find_root, func_lib ) ]
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
TVF FUNCTION func_lib [/*

    proc find_root { W L TA } {
      set a [$TA] ; # target area
      set l [$L]  ; # length
      set w [$W]  ; # width

      set b [expr -($l + $w)]; #  b in the quadratic formula
      # Discriminant of quadratic formula:
      set discr [expr sqrt(${b}*$b - 4*($l*$w - $a))]
      set root1 [expr (-$b - $discr)/2]
      # We want the solution that is less than the width
      if {($root1 < $w) && ($root1 > 0)} {
           return $root1
      }
      set root2 [expr (-$b + $discr)/2]
      if {($root2 < $w) && ($root2 > 0)} {
       return $root2
      }
      puts "---NOTE: QUAD_ROOT returned 0."
      return 0
    }
  */]
```

3. Use DFM Property to find the resizing value *size_val* for each polygon on layer L0 and attach it to each polygon as a DFM property. The *size_val* is half of *shrink_val*.

```
LAYER L0 0

// Call QUAD_ROOT function to solve for shrink_val.
// DFM SIZE needs one-half of shrink_val.
VARIABLE scale 0.3  // scale factor should be less than one.
L0_prop  = DFM PROPERTY L0
              [weff = W_EFF(L0)]  // effective width
              [leff = L_EFF(L0)]  // effective length
              [target_area = scale*area(L0)]
              [size_val = (-0.5)*QUAD_ROOT(PROPERTY_REF(weff),
                                           PROPERTY_REF(leff),
                                           PROPERTY_REF(target_area))]
```

4. Resize the polygons with DFM Size and output the layer to the standard DRC results database.

```
sized_L0 = DFM SIZE L0_prop BY "size_val"
sized_L0 { COPY sized_L0 }
```

5. Write a check to examine the results. Compare the actual area to the target area and check that the original and final edge counts are the same.

   Edge counts are compared because it is possible for an edge to be removed if two times the *size_val* is greater than or equal to the smallest edge length of the polygon; see the results section for an example.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

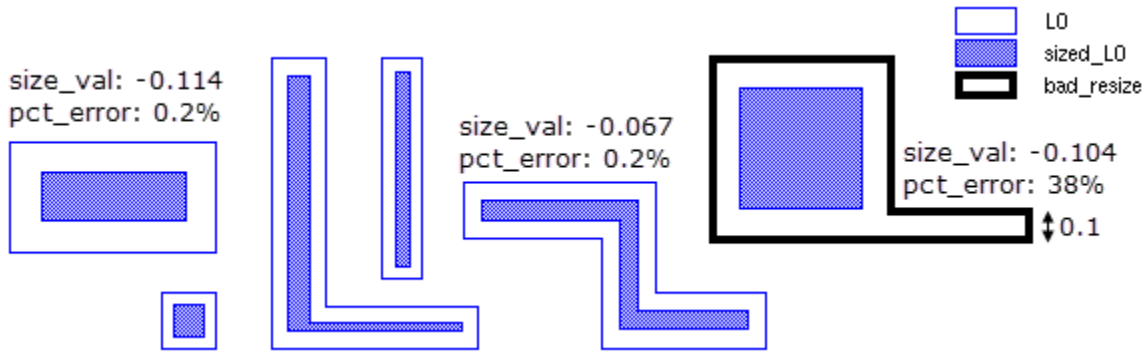
```
// Error if percent difference between actual and expected area
//   is greater than pct_err_max.
// Error if original and resized shape do not have same edge count.
// Return original shape for error conditions.
// Note: measurement functions on size_L0 layer sum over secondary
//   layer shapes, but there is only one shape in each cluster.
VARIABLE pct_err_max 10
bad_resize = DFM PROPERTY L0_prop sized_L0 OVERLAP
               [ pct_error = 100*ABS(AREA(sized_L0) -
                                 PROPERTY(L0_prop, target_area))
                                /PROPERTY(L0_prop, target_area)]
               [ edge_diff = COUNT(L0_prop:EDGE) -
                                COUNT(sized_L0:EDGE) ]
               [ - = ( (PROPERTY_REF(edge_diff) != 0) ||
                        (PROPERTY_REF(pct_error) > pct_err_max) )
                        ? 1 : 0 ] == 1
```

6. Output the results to a DFM RDB:

```
// Set default options for DFM RDB operations
DFM DEFAULTS RDB CELL SPACE ALL CELLS CHECKNAME "%_l_"
// Output the results
size_by_check {
    DFM RDB bad_resize eqdrc.rdb
        COMMENT "Original shapes where resized shape has different "
        COMMENT "number of edges, or difference between target and "
        COMMENT "actual new size is greater than ^pct_err_max %"
//for debug
DFM RDB L0_prop eqdrc.rdb
}
```

Results

The following figure shows results for several polygons. The values of size_val and pct_error are shown for several of the polygons.



For the upper right polygon, twice the *size_val* is greater than the minimum edge length of 0.1, so that edge is removed during resizing. The bad_resize check outputs this shape as an error because the original and resized edge counts are not the same. The percentage error between the final and target area is less than 2% for all of the other polygons.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Depending on your application, you can add special error handling to the L0_prop derivation or QUAD_ROOT function to catch cases where twice the *size_val* is greater than or equal to the minimum edge length.

Related Topics

Introduction to DFM Property [Calibre Verification User's Manual]

Introduction to DFM RDB [Calibre Verification User's Manual]

Measurement Functions [Standard Verification Rule Format (SVRF) Manual]

Supported Math Functions in DFM Expressions [Standard Verification Rule Format (SVRF) Manual]

Math Operators [Standard Verification Rule Format (SVRF) Manual]

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Chapter 3
Chip-Level Verification

Chip-level verification for recent design nodes has become steadily more complex and time-consuming. DRC results counts have increased geometrically and LVS errors are difficult to debug. The Calibre chip-level verification tools provide a way to run quick turn-around verification at an early stage in the design process. Finding and fixing errors at an early stage in the design process can shorten the overall physical verification time.

Calibre nmDRC Reconnaissance and DRC Analyze Modes

Chip-level verification for recent design nodes has resulted in DRC results counts increasing geometrically. This necessitates identifying regions of concern in a layout early in the physical design cycle. Calibre supports two run modes for addressing this issue: Calibre nmDRC Reconnaissance mode and DRC Analyze mode. To use Calibre nmDRC Recon or DRC Analyze, you need a layout, a rule file (typically a foundry deck), and a Calibre RealTime Digital license. No additional configuration of an existing Calibre nmDRC verification environment is needed.

The Calibre nmDRC Recon tool performs automated rule check selection that provides fast feedback to help designers identify systemic design issues that may result in many errors. A Calibre nmDRC Recon run primarily excludes context-dependent rules, such as connectivity, delta v, and multi-patterning rule checks, which are likely to result in long run times and are unlikely to provide actionable results for an unfinished layout. The excluded rule checks require significant context awareness and usually generate hundreds or thousands of violations at different locations across the chip; usually such violations are resolved indirectly as the design matures. By contrast, a Calibre nmDRC Recon run is fast and identifies results that are typically closely related to the source of the design flaw. The faster and more targeted Calibre nmDRC Recon run enables you to identify and fix significant issues in your layout before performing sign-off verification runs.

Calibre nmDRC Recon mode is intended for full-chip runs, but it can also be used on blocks. Simply specify -recon on the hierarchical DRC command line—no special rule file configuration needs to occur. This mode is designed to give a general idea of how “dirty” the design is. Debugging of results is the same as for other DRC runs.

You can customize the set of checks that are run by Calibre nmDRC Recon. As the design becomes more complete, you can gradually add more rule checks. See “Selecting Checks in Calibre nmDRC Recon” on page 95.

Calibre DRC Analyze is a run mode intended for full-chip DRC error distribution analysis. Simply specify -analyze on the hierarchical DRC command line to use this mode. This functionality produces a DFM database containing rule check results that allow you to

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

quickly determine areas of a chip requiring the most attention to enhance layout quality. Using -analyze can have a minor performance impact.

DRC Analyze results support the construction of histograms and colormaps to see rule violation “hotspots” in the design. The results are similar to those seen in a DFM Report Card or a Density RDB. The results can be analyzed in Calibre RVE by layer, rule check, check set, cell, and data capture window.

Calibre nmLVS Reconnaissance Short Isolation and Soft Connection Checking

Calibre nmLVS Recon SI provides stand-alone layout short isolation. This short isolation is performed independently of an LVS run. Calibre nmLVS Recon SI runs have proven to be very fast, and using this flow early in the design verification process can prevent various LVS problems later on.

There are two primary short isolation flows: the rule file flow and the Mask SVDB Directory flow. The rule file flow performs short isolation using LVS circuit extraction rules and a layout as inputs. The Mask SVDB Directory flow loads an SVDB from a previous LVS circuit extraction run or a previous Calibre nmLVS Recon SI run, and performs short isolation on that data. Only connectivity extraction and short isolation are performed (no device extraction or anything dependent upon it).

Calibre nmLVS Recon Softchk provides stand-alone soft-connection checking similar to LVS Softchk. Like Calibre nmLVS Recon SI, Calibre nmLVS Recon Softchk is performed independently of an LVS run. Calibre nmLVS Recon Softchk is only performed on a Mask SVDB Directory from a circuit extraction run.

These stand-alone runs require a Calibre nmLVS Reconnaissance license and a Calibre nmLVS-H license pair. Depending on the flow used, a Calibre Query Server license may also be required.

Running Calibre nmDRC Recon 94
Selecting Checks in Calibre nmDRC Recon 95
Running Calibre DRC Analyze 98
Calibre RVE Feature Overview 99
Reviewing DRC Analyze Results 100
Running Calibre nmLVS Recon SI 107
Running Calibre nmLVS Recon Softchk. 109

Running Calibre nmDRC Recon

Run Calibre nmDRC Reconnaissance mode to automatically execute a subset of rule checks in a rule file.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Prerequisites

- A layout design.
- A rule file, typically a foundry deck.
- A Calibre RealTime Digital license and sufficient Calibre nmDRC/Calibre nmDRC-H license pairs for the run configuration.

Procedure

1. (Optional.) For portions of the design you want to ignore, specify either of the following:

   ```
   LAYOUT WINDEL x1 y1 x2 y2
   ...
   LAYOUT WINDEL CELL my_cell ORIGINAL
   ...
   ```

2. (Optional.) Specify DRC Recon Add Check, DRC Recon Remove Check, DRC Recon Add Check By Layer, and DRC Recon Remove Check By Layer statements in the rule file to control the check set for the run.

   See "Selecting Checks in Calibre nmDRC Recon" on page 95 for an example.

3. To run Calibre nmDRC Recon mode, use the following command line (csh assumed):

   ```
   % calibre -drc -hier -turbo -recon rules >&! log
   ```

   Include the -hyper and -remote or -remotefile options as supported by your hardware. To run the set of checks that -recon does not run, specify "-recon inverse".

   The -recon run options are also available from the Calibre Interactive nmDRC **Inputs** pane.

Results

The output files include the run transcript, a DRC Summary Report if specified, and a DRC Results Database. The database can be opened with Calibre RVE for cross-probing results into your layout editor.

Related Topics

Running Calibre DRC Analyze

Selecting Checks in Calibre nmDRC Recon

Selecting Checks in Calibre nmDRC Recon

By default, Calibre nmDRC Recon automatically enables a set of rule checks that run fast and are not context-dependent. You can customize the checks that are run, depending on where you are in the design cycle.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

These specification statements assist in customizing the checks that are enabled:

Specification Statement	Description
DRC Recon Add Check	Executes only the specified checks. You can specify predefined groups of checks with keywords. For example, RECON_CONNECT selects all connectivity checks.
DRC Recon Remove Check	Removes the specified checks. You can specify predefined groups of checks with keywords.
DRC Recon Add Check By Layer	Selects checks to execute by layer.
DRC Recon Remove Check By Layer	Selects checks to omit by layer.

When using these statements, an explicitly named rule check must be selected to run in a standard Calibre nmDRC run. All of the add statements are processed first, then the remove statements, so it is possible for a remove statement to disable a check that is enabled by an add statement. See the command references for more detail.

The following table lists the keywords for the predefined rule check groups available with the DRC Recon Add Check and DRC Recon Remove Check statements. The Default column indicates whether the set of rule checks is executed in a default Calibre nmDRC Recon run.

Table 3-1. Keywords for Rule Check Groups in Calibre nmDRC Recon

Keyword	Definition	Default?
RECON_DEFAULT	All rule checks that would be executed by default in Calibre nmDRC Recon mode.	Yes
RECON_DENSITY	All selected rule checks that require a Density operation.	Yes
RECON_OFFGRID	All selected rule checks that require an Offgrid Directional or non-simple Offgrid operation. A non-simple Offgrid operation uses any of the following keywords: INSIDE OF LAYER, RELATIVE, ABSOLUTE, EDGE, HINT, or OFFSET.	No
RECON_CONNECT	All selected rule checks that require connectivity, including those with DFM DV operations.	No
RECON_DV	All selected rule checks that require a DFM DV operation.	No
RECON_MP	All selected rule checks that require a Calibre Multi-Patterning operation.	No
RECON_PM	All selected rule checks that require a Calibre Pattern Matching operation.	No

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following procedure gives an example of how to use these statements to customize the set the checks that are run by Calibre nmDRC Recon, depending on what stage in the design cycle you are in.

Procedure

1. For the initial run, execute the default set of Calibre nmDRC Recon checks, but exclude all density checks and exclude all checks that depend on layers m3, m4, and m5 anywhere in the layer derivation tree of the rule check. Add this to the rule file:

   ```
   DRC RECON REMOVE CHECK BY LAYER m3 m4 m5
   DRC RECON REMOVE CHECK RECON_DENSITY
   ```

   Start a run:

   ```
   calibre -drc -hier -turbo -recon rules |& tee log
   ```

   Debug the results and proceed to the next stage when the design is ready.

   Tip

   If you use DRC Recon Add Check By Layer, it adds *all* checks that require the specified layer, including checks not selected by default in a Calibre nmDRC Recon run. To remove checks not run by default, do the following:

   ```
   DRC RECON ADD CHECK BY LAYER m1
   DRC RECON REMOVE CHECK RECON_CONNECT RECON_DV RECON_MP
             RECON_PM RECON_OFFGRID
   ```

2. Run with the default set of Calibre nmDRC Recon checks on all layers, but still exclude density checks. Comment out the first specification statement from the previous step:

   ```
   //DRC RECON REMOVE CHECK BY LAYER m3 m4 m5
   DRC RECON REMOVE CHECK RECON_DENSITY
   ```

   Do a Calibre nmDRC Recon run and debug the results. Proceed to the next stage when the design is ready.

3. Run with the default set of Calibre nmDRC Recon checks, include density checks and connectivity checks, but exclude DFM DV checks. Density checks are run by default, so it is not necessary to explicitly add them.

   ```
   DRC RECON ADD CHECK RECON_DEFAULT RECON_CONNECT
   DRC RECON REMOVE CHECK RECON_DV
   ```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

If you use DRC Recon Add Check or DRC Recon Add Check By Layer, you must specify all checks or rule check groups that you want to execute. For example, the following statement executes *only* the connectivity checks:

```
DRC RECON ADD CHECK RECON_CONNECT // does not include default checks
```

Do a Calibre nmDRC Recon run and debug the results. Continue this process as necessary. The remaining checks to enable are DFM DV, Calibre Pattern Matching, and Calibre Multi-Patterning checks.

Related Topics

Running Calibre nmDRC Recon

Running Calibre DRC Analyze

Run Calibre DRC Analyze to get error distribution results data for gross analysis of areas of a design that require attention.

Prerequisites

- A layout design.
- A rule file, typically a foundry deck.
- A Calibre RealTime Digital license and sufficient Calibre DRC/Calibre DRC-H license pairs for the run configuration.

Procedure

1. (Optional.) For portions of the design you might wish to ignore, specify either of the following:

   ```
   LAYOUT WINDEL x1 y1 x2 y2
   ...
   LAYOUT WINDEL CELL my_cell ORIGINAL
   ...
   ```

2. To run the complete set of rule checks, use the following command line (csh assumed):

   ```
   % calibre -drc -hier -turbo -analyze rules >&! log
   ```

   Include the -hyper and -remote or -remotefile options as supported by your hardware.

3. (Optional.) To run a more fundamental set of checks, add the -recon option:

   ```
   % calibre -drc -hier -turbo -analyze -recon rules >&! log
   ```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Results

The output files include the run transcript, a DFM Summary Report called *dfm.rep*, and a DFM Database called *analyze.dfmdb*. The database can be opened with Calibre RVE for cross-probing results into your layout editor.

A DRC Summary Report is not produced; see the *dfm.rep* file instead. The DRC Results Database is empty. The *analyze.dfmdb* database contains all results.

Related Topics

Running Calibre nmDRC Recon

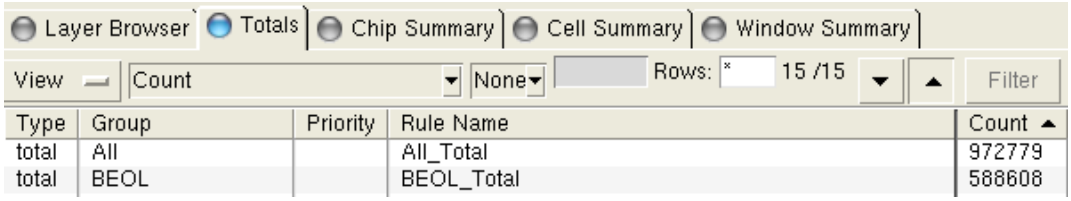
Calibre RVE Feature Overview

Reviewing DRC Analyze Results

Calibre RVE Feature Overview

Calibre RVE for DFM is used to access the DRC Analyze results. The functionality is similar to that in a DFM Report Card. The main difference is, DFM Analyze data is concerned with rule check results counts and not other statistics in a DFM Report Card.

When you open the *analyze.dfmdb* file in Calibre RVE, there is a series of tabs like this:



The tabs contain the following information. (Links are to the *Calibre YieldAnalyzer and YieldEnhancer User's and Reference Manual* and the *Calibre YieldServer Reference Manual*.)

Table 3-2. Calibre RVE for DFM Results Tabs

Tab	Contents	Feature Details
Layer Browser	Original and derived layer names, rule check comments, and layer operations that produce layers.	Using a DFM Database in Calibre RVE Layer Browser Tab
Totals	Results arranged by rule check group with aggregate result counts for all checks in the group	Totals[1]
Chip Summary	Detailed presentation of the **Totals** tab data broken down by rule check.	Chip Summary[1]
Cell Summary	Cell names, areas, instance counts, and hierarchical results counts.	Cell Summary[1]

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 3-2. Calibre RVE for DFM Results Tabs (cont.)

Tab	Contents	Feature Details
Window Summary	Data capture window indices with result counts per window for each rule check.	Window Summary[1]

1. Most DRC Analyze tabs are simplified versions of the DFM Report Card tabs containing only rule check result counts rather than other DFM statistics.

Every tab except for **Layer Browser** contains a menu bar at the top. The fields in the menu bar control what appears in a table and are discussed under the DFM Report Card "Overview" section of the *Calibre YieldAnalyzer and YieldEnhancer User's and Reference Manual*.

Related Topics

Running Calibre DRC Analyze

Reviewing DRC Analyze Results

Reviewing DRC Analyze Results

Calibre RVE for DFM allows review of DRC Analyze results. The presentation is similar to DFM Report Card and Density RDB results. This topic shows some of the more commonly used features.

Prerequisites

- A layout design and viewer.
- An *analyze.dfmdb* results database.
- A Calibre RVE license.

Procedure

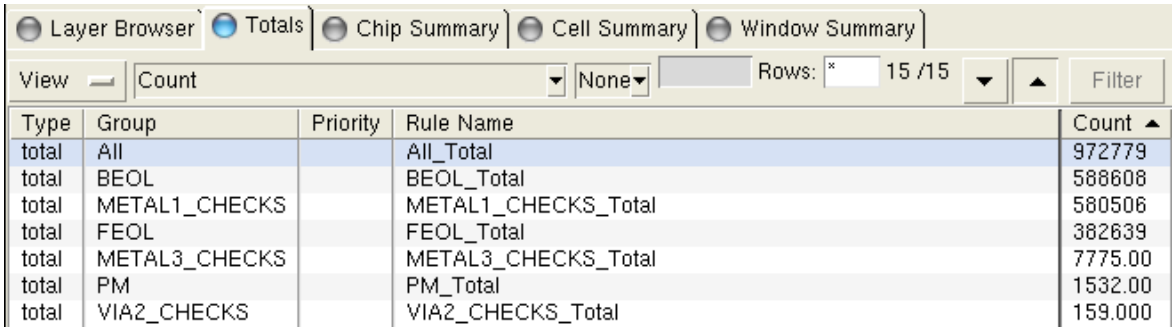
1. Open the design and Calibre RVE using your layout viewer.
2. Open *analyze.dfmdb* in Calibre RVE using **File > Open Database**. Ensure the **Database Type** is DFM.

   (Optional.) This command line can be used to start the review session in Calibre DESIGNrev:

   ```
   calibredrv design.oas -rve -dfm analyze.dfmdb
   ```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Calibre RVE opens with the **Totals** tab selected:

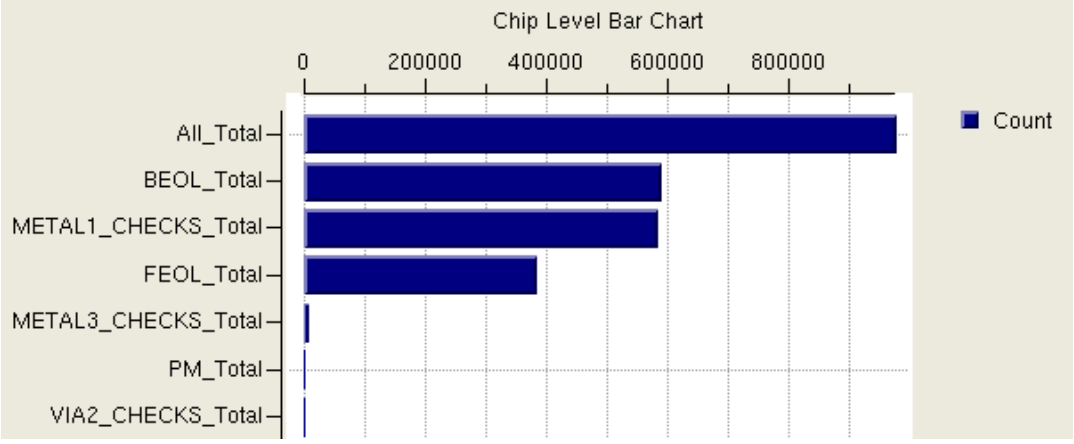


Type	Group	Priority	Rule Name	Count
total	All		All_Total	972779
total	BEOL		BEOL_Total	588608
total	METAL1_CHECKS		METAL1_CHECKS_Total	580506
total	FEOL		FEOL_Total	382639
total	METAL3_CHECKS		METAL3_CHECKS_Total	7775.00
total	PM		PM_Total	1532.00
total	VIA2_CHECKS		VIA2_CHECKS_Total	159.000

The data is organized by rule check group in descending order of total result count.

3. In the **Totals** tab, select **View > Bar Chart**.

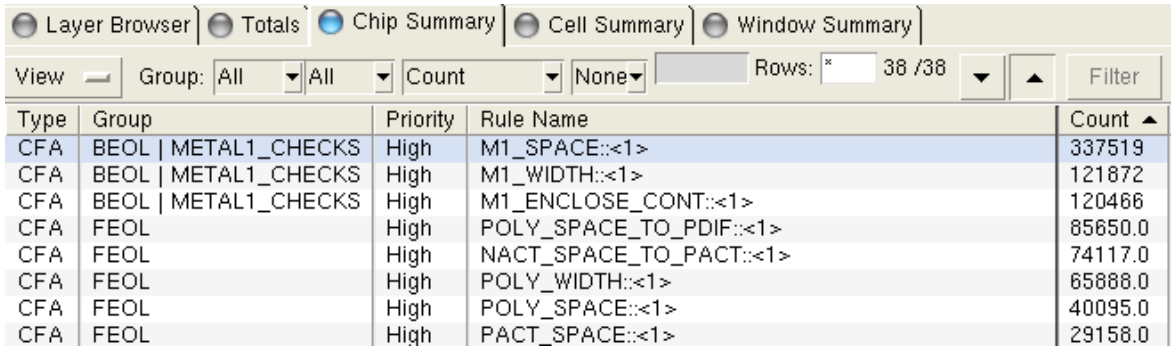
   The data is presented in bar chart form:



4. In the **Totals** tab, select **View > Table** to restore the tabular form.

   The table and bar chart data viewing modes are available in all tabs with rule check statistics.

5. Select the **Chip Summary** tab.

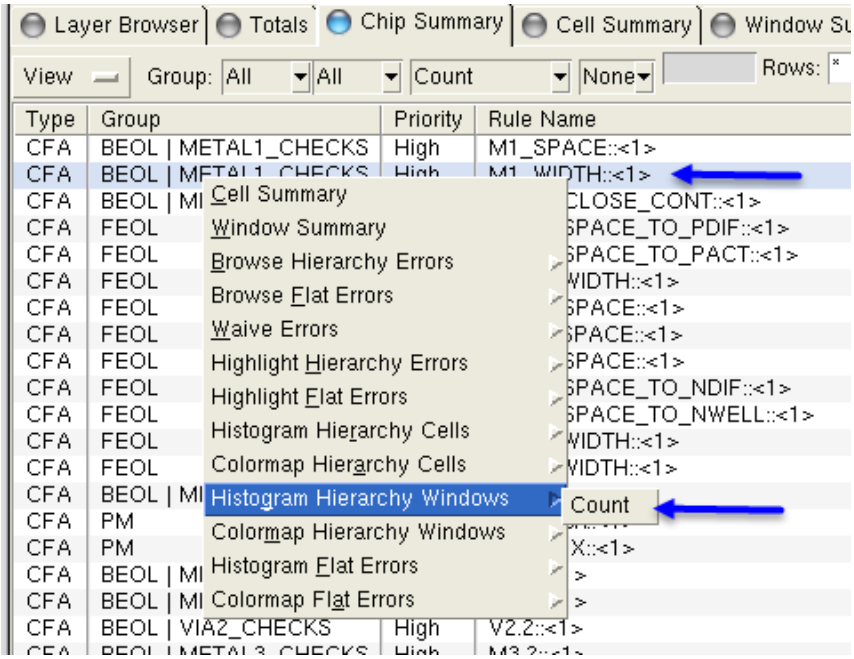


Type	Group	Priority	Rule Name	Count
CFA	BEOL | METAL1_CHECKS	High	M1_SPACE::<1>	337519
CFA	BEOL | METAL1_CHECKS	High	M1_WIDTH::<1>	121872
CFA	BEOL | METAL1_CHECKS	High	M1_ENCLOSE_CONT::<1>	120466
CFA	FEOL	High	POLY_SPACE_TO_PDIF::<1>	85650.0
CFA	FEOL	High	NACT_SPACE_TO_PACT::<1>	74117.0
CFA	FEOL	High	POLY_WIDTH::<1>	65888.0
CFA	FEOL	High	POLY_SPACE::<1>	40095.0
CFA	FEOL	High	PACT_SPACE::<1>	29158.0

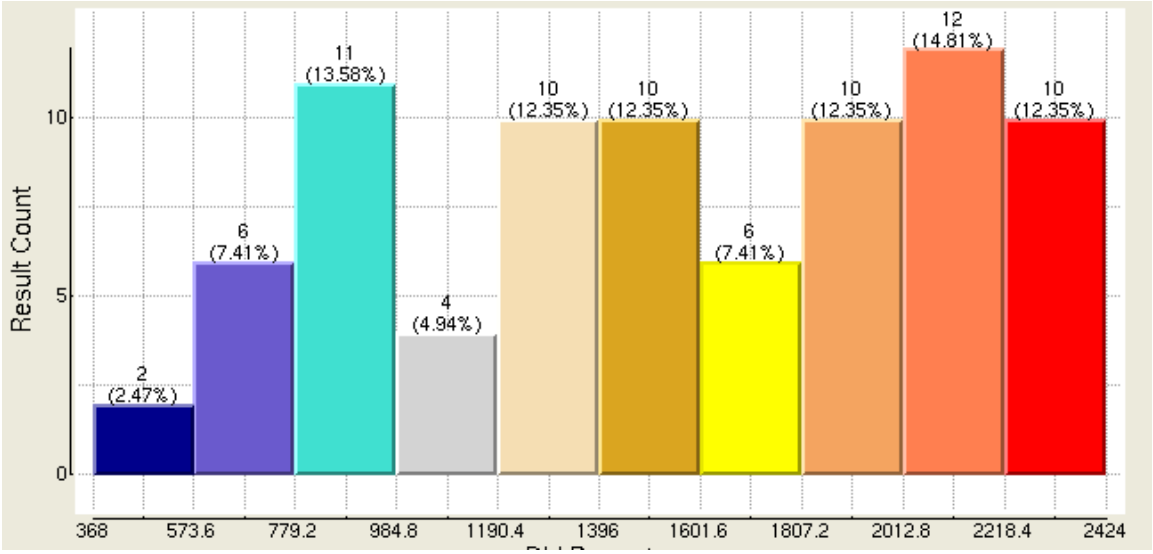
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The data is organized by rule check name in descending order of result count. The rule check group names are also displayed.

6. In the **Chip Summary** tab, choose a rule check name, right-click on it, and select **Histogram Hierarchy Windows > Count**.



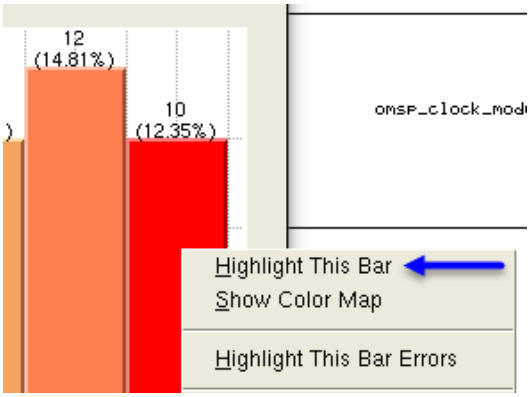
A panel opens with the rule check count data arranged by histogram bin:



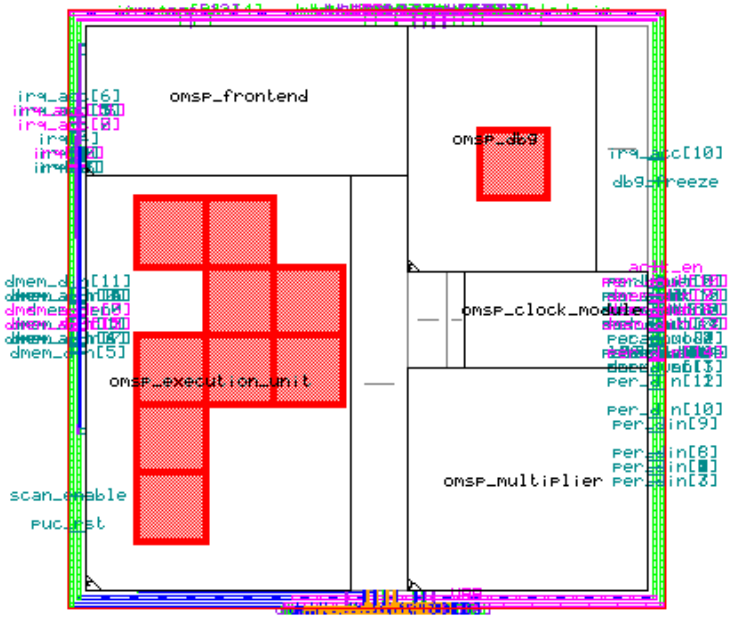
The bins comprise result count intervals of increasing values from left to right. Each bin corresponds to a set of data capture windows in the layout extent.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

7. In the histogram panel, right click in one of the bars and select **Highlight This Bar**.



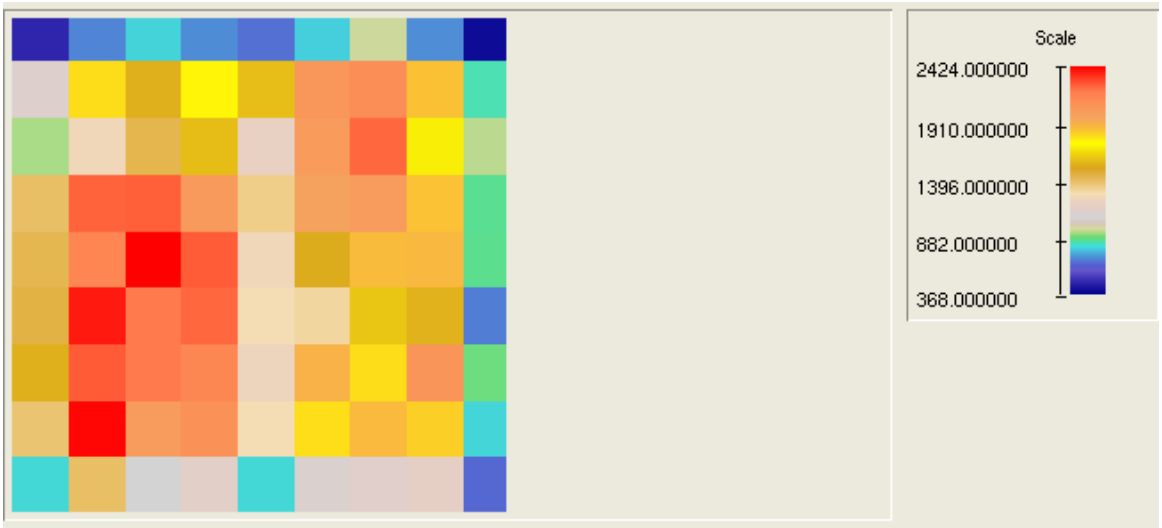
The corresponding data capture windows highlight with the same color in the layout:



8. Repeat step 6 using the **Colormap Hierarchy Windows > Count** menu item instead.

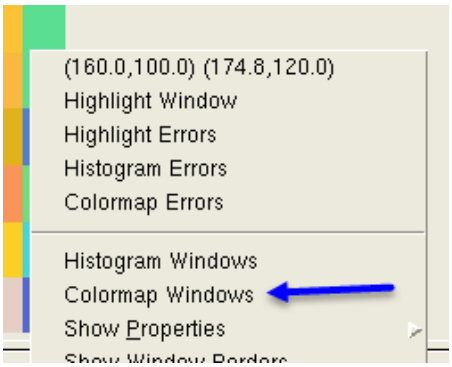
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

A panel opens with the rule check count data arranged by data capture window:



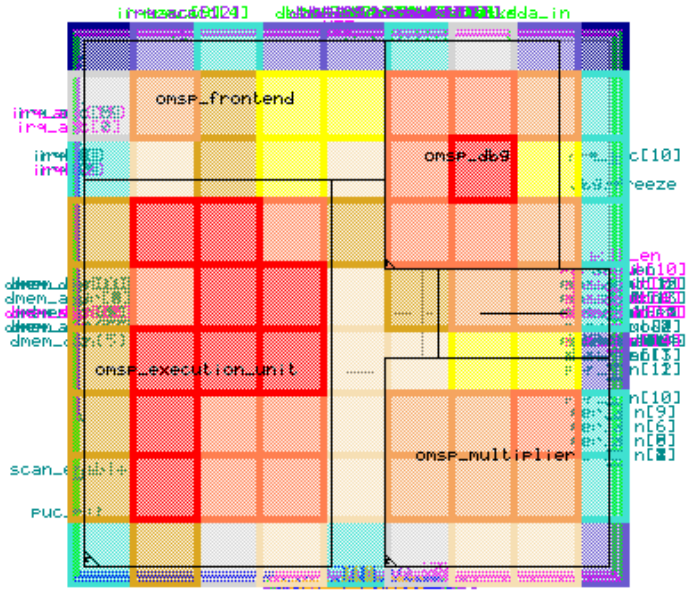
The Scale represents results count per window in an increasing “temperature” color spectrum.

9. In the colormap panel, right-click and select **Colormap Windows**.

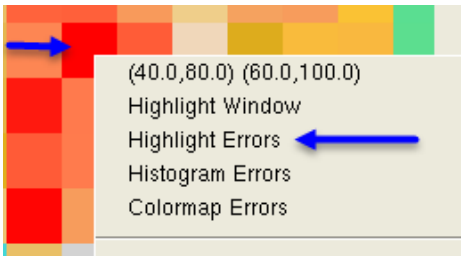


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

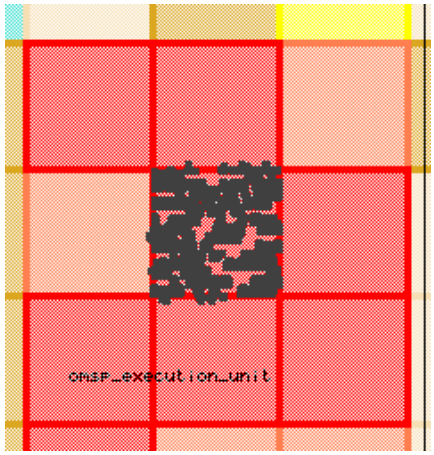
The data capture windows overlay the layout using the colormap pattern:



10. Choose one of the data capture window boxes in the colormap panel. Right-click in that box and select **Highlight Errors**.



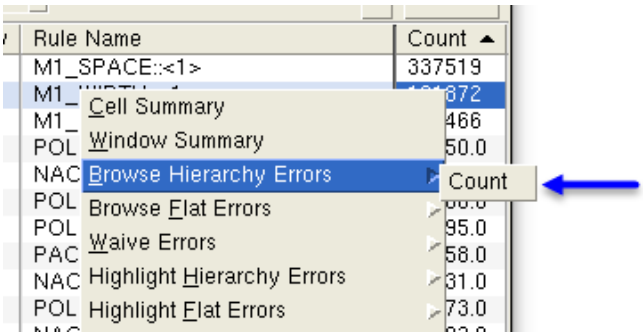
The corresponding rule check errors highlight in the layout:



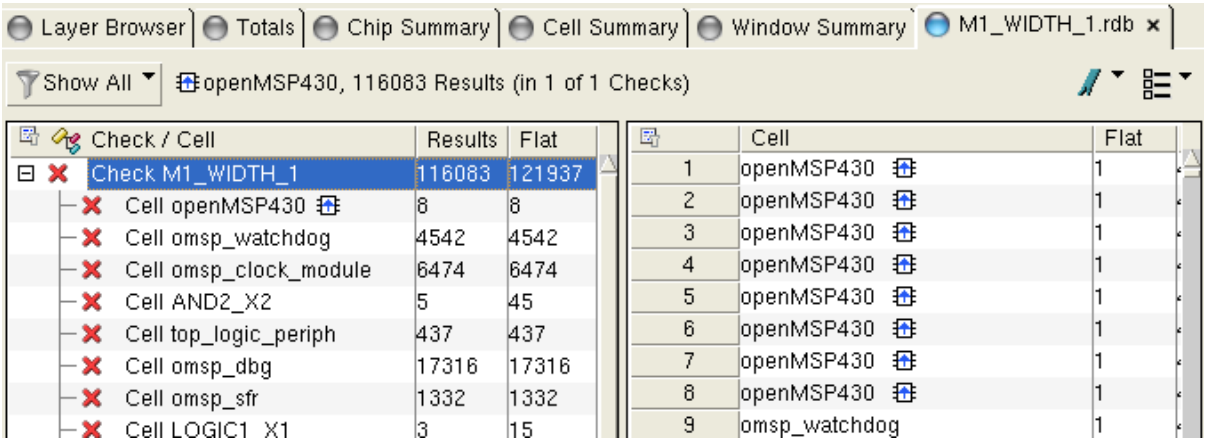
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Zoom in to the error region using your layout viewer control to see the details.

11. In the **Chip Summary** tab, choose a rule check name, right-click on it, and select **Browse Hierarchy Errors > Count**.

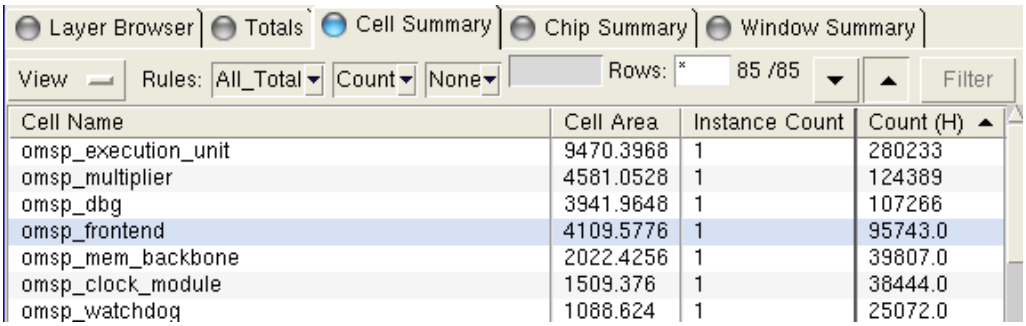


A ***<rule_check_name>*.rdb** tab opens:



This is the usual ASCII RDB presentation for DRC results. You can use this to highlight individual results or groups of results in the layout.

12. In Calibre RVE, select the **Cell Summary** tab.



The data is arranged by cell name in decreasing order of results count. Cell area in um^2 and instance count are given.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

Running Calibre DRC Analyze

Calibre RVE Feature Overview

Running Calibre nmLVS Recon SI

Calibre nmLVS Recon SI performs stand-alone short isolation independent of an LVS run. Such runs are typically very fast and can be used early in the design verification process to avoid various problems in later LVS runs.

There are two flows:

- Rule file flow — Performs short isolation using LVS circuit extraction rules and a layout as inputs.
- Mask SVDB Directory flow — Loads an SVDB from a previous LVS circuit extraction run or previous Calibre nmLVS Recon SI run, and performs short isolation on that data.

Only connectivity extraction and short isolation are performed (no device extraction or anything dependent upon it). For complete details, see "-recon -si" in the "Calibre nmLVS Reconnaissance Command Line" section of the *Calibre Verification User's Manual*.

Prerequisites

- A Calibre nmLVS Reconnaissance license.

  If -si=*tcl_script* is specified, a Calibre Query Server license is also required.

Required inputs depend on the flow:

- Rule file flow:
  - A layout design.
  - A rule file with LVS circuit extraction rules that include a Mask SVDB Directory statement that produces a persistent hierarchical database (PHDB). Some of the common keyword options produce the PHDB are include CCI, PHDB, QUERY, and SI.
  - A Calibre nmLVS/Calibre nmLVS-H license pair.
- Mask SVDB Directory flow:
  - An SVDB from a previous LVS circuit extraction run in which the Mask SVDB Directory SI keyword was used or a previous Calibre nmLVS Recon SI run.

Procedure

1. (Optional) Specify the Connect and Sconnect statements that are processed prior to short isolation. By default, all such statements that produce connectivity to the level of the

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

device pins are processed. Reducing the number of Connect statements that are processed improves runtime at the potential cost of finding fewer shorts.

The method you use depends on the flow:

- Rule file flow: Include the LVS SI Select Connects statement in the rule file.
- Either flow: Specify a Query Server Tcl shell script with the short_db::select_connects command. The Query Server script is specified on the command line in the next step.

2. Run Calibre nmLVS Recon SI with one of the following command line syntaxes, where *opts* corresponds to run options such as -turbo:

   Rule file flow:

   calibre -recon -si['='{ALL | DB | IO | PG | *tcl_script* }] [*opts*] ***rule_file_name***

   Mask SVDB Directory flow

   calibre -recon -si['='{ALL | IO | PG | *tcl_script* }] [*opts*] **-svdb *svdb_dir*** [*top_cell*]

   By default, all nets at the primary level of the design are processed for short isolation. If the optional equals sign (=) is specified after **-si**, then the ALL, IO, and PG modal options specify to process nets at all levels, I/O signal nets at the primary level, or power and ground supply nets at the primary level. The *tcl_script* option specifies a Query Server Tcl shell script that runs commands from that interface on the PHDB data. The DB modal option is used for two-step processing where just the Mask SVDB Directory is produced in the first run, and short isolation is run separately on the SVDB in subsequent runs. This can have performance benefits in some cases.

   An LVS Isolate Shorts NO statement, specified either explicitly or by default, is ignored in this flow. LVS Isolation Shorts YES secondary keywords are observed if specified in the rules and are applied to the run if they do not conflict with the command line option behavior. See the "-recon -si" command-line argument descriptions in the *Calibre Verification User's Manual* for further details.

Results

Connectivity text shorts, opens, and unattached label warnings are given in the transcript, which is the same as for short isolation when performed in a calibre -spice run. However, a circuit extraction report is not written, nor is a *.shorts* database.

The tail of the transcript contains a section like this when shorts are detected:

```
Current short serial number is 1
  Short circuit (index: 0) (cell: TOPCELL) (net_id: 2) (text: PWR).
    PWR at (1.3 82.4) (layer metal2)
    top_short at (3.4 78) (layer metal2)

Warning: 1 short paths from 1 out of 1 shorts were selected in primary
cell TOPCELL for short isolation.
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Shorts from non-primary cells may also be shown depending on rule file settings. If no shorts are detected, then this is shown instead:

```
Short Isolation results are clean for ALL cells.
```

The rule file flow creates a new Mask SVDB Directory with short isolation results, while the Mask SVDB Directory flow updates the existing SVDB directory. Short isolation results from either flow can be viewed and debugged as usual in Calibre RVE for LVS. The results presentation follows the LVS Isolate Short BY CELL BY LAYER options. You can also use the Calibre Query Server Tcl shell.

Related Topics

Mask SVDB Directory [Standard Verification Rule Format (SVRF) Manual]

LVS Isolate Shorts [Standard Verification Rule Format (SVRF) Manual]

Interactive Short Isolation [Calibre RVE User's Manual]

Using Short Database Commands [Calibre Query Server Manual]

Running Calibre nmLVS Recon Softchk

Calibre nmLVS Recon Softchk performs stand-alone Sconnect stamping conflict (soft connection) checks independent of an LVS run. This permits conducting these checks in parallel with things like stand-alone short isolation.

These verification runs are performed on a Mask SVDB Directory produced with the SI keyword. For complete details, see "-recon -softchk" in the "Calibre nmLVS Reconnaissance Command Line" section of the *Calibre Verification User's Manual*.

Prerequisites

- A Calibre nmLVS Reconnaissance license.
- Calibre nmLVS/Calibre nmLVS-H license pairs.

  A Calibre Query Server license is also required if using -softchk=*tcl_script*.
- A layout design.
- An LVS circuit extraction rule file.

Procedure

1. Specify the SI keyword in a Mask SVDB Directory statement in your rule file.
2. Run hierarchical circuit extraction:

   ```
   calibre -spice layout.sp -turbo ... rules |&tee log
   ```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

3. Run stand-alone soft connection checks:

```
calibre -recon -softchk -turbo -svdb svdb
```

4. (Optional.) As an alternative to step 3, prepare a Query Server Tcl shell script called *softchk.qs* that contains either the qs::softchks or qs::softchk command. Then run stand-alone soft connection checks as follows:

```
calibre -recon -softchk=softchk.qs -turbo -svdb svdb
```

Results

If step 3 is followed, soft connection checks are run on the layout data in the PHDB in a similar manner as LVS Softchk LOWER in a circuit extraction run. The checks are run on any Sconnect stamped layers (the second layer in the Sconnect syntax) for which there was a stamping conflict during circuit extraction. Such conflicts are indicated in the circuit extraction report like this:

```
WARNING:  Stamping conflict in SCONNECT - Multiple source nets stamp one
target net.
```

If step 4 is followed, then soft connection checks are run on the layout data in the PHDB according to the qs::softchks or qs::softchk command configuration.

The output filename is *softchk.rdb*, and this can be opened as an ERC database in Calibre RVE.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Chapter 4
Density Best Practices

The Density operation is typically used for checking the ratio of a layer's area per unit area of a chip design. This chapter focuses on recommended ways to implement Density checks and analyze their output.

Analyzing Density Results with Calibre RVE 111
Density Operation Categories 117

Analyzing Density Results with Calibre RVE

Visualizing Density results using Calibre RVE is important for interpreting what they mean and how to fix them. Results visualization also assists you in writing better Density checks by providing properties that can be sorted, highlighted in color maps and histograms, and displayed in the layout itself.

Prerequisites

- Calibre nmDRC and Calibre RVE licenses.
- DRC rule file and geometric layout.
- Basic understanding of the Density layer operation.
- Familiarity with using Calibre RVE. See "Using Calibre RVE" in the *Calibre RVE User's Manual*.
- Integration between your layout viewer and Calibre RVE. See "Connection to a Design Tool" in the *Calibre RVE User's Manual*.

Procedure

1. Open your rule file and add the RDB option to a Density layer operation. For example:

```
filldensity3 = DENSITY fill keepout < d
  [(AREA(fill)/!AREA(keepout))/AREA()] WINDOW w STEP s
  RDB filldensity3.rdb

filldensity3 {
@ Density fill< ^d with function (AREA(fill)/!AREA(keepout))/AREA()
COPY filldensity3
}
```

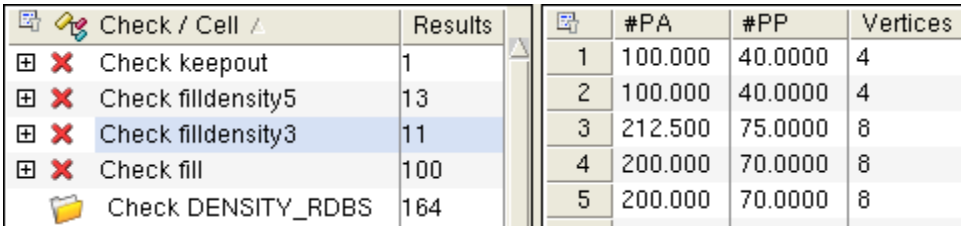
   The layer operation's constraint, window, and step parameters should reflect something useful for your process. The RDB file *filldensity3.rdb* is used for detailed results analysis in Calibre RVE.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

2. Run DRC in the usual way for your flow using the rule from Step 1. Use the DRC Select Check statement in the rule file as necessary to limit results output.

3. If it is not already open, start your layout viewer and open your layout.

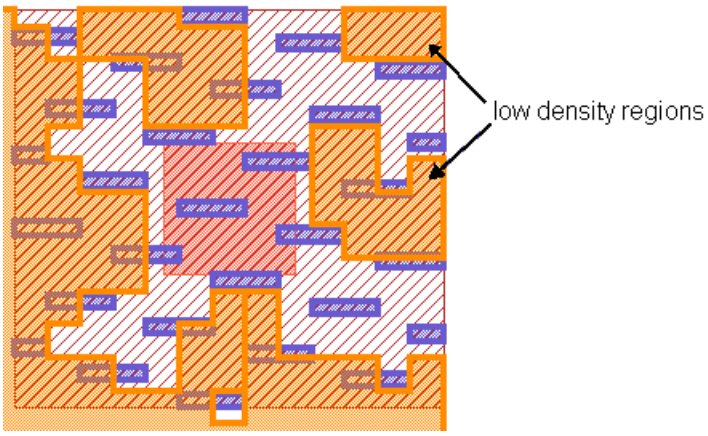
4. Start Calibre RVE in the usual way for your flow and open your DRC Results Database. The results for the filldensity3 rule check shown in Step 1 might appear as follows in Calibre RVE:



Check / Cell	Results
Check keepout	1
Check filldensity5	13
Check filldensity3	11
Check fill	100
Check DENSITY_RDBS	164

	#PA	#PP	Vertices
1	100.000	40.0000	4
2	100.000	40.0000	4
3	212.500	75.0000	8
4	200.000	70.0000	8
5	200.000	70.0000	8

5. Highlight the filldensity3 results by selecting **H** on your keyboard. The results could appear like this:



The results from the DRC Results database show merged regions of low density. While informative, this may not be enough detail for you to decide what the results mean. The following steps show how to get more detail.

6. In Calibre RVE, select **Setup** > **Options**. In the Options tab, ensure the following settings are enabled:

   DRC Browser — Properties: **Show calculated properties**; Side RDBs: **Density**: **Show As Check**

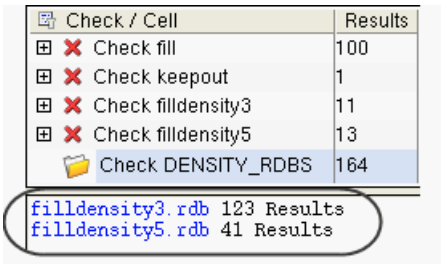
   Highlighting — Layout Highlighting: Highlight Text Size: **0.2**

   DRC/DFM Highlighting: **Show Check Names**, **Show Result IDs**, **Show Properties: ALL**.

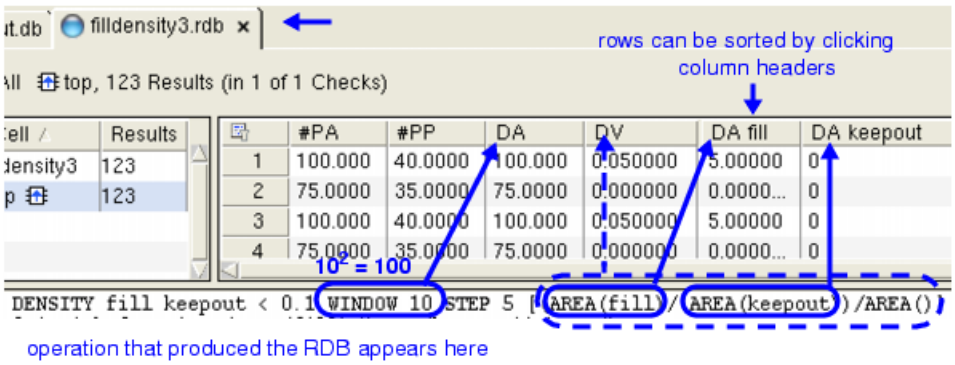
7. Select **Apply** to save your settings and **Close** to dismiss the Options tab.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

8. Ensure the **View > Check Text Pane** option is selected. Calibre RVE then displays the Density RDB files as a DENSITY_RDBS rule check with RDB files listed as links in the check text pane:



9. Select the filldensity3.rdb link. The RDB results file opens in a new tab.



The results table shows the following columns by default. Areas are in square um.

— Result number.

#PA — Result polygon area.

#PP — Result polygon perimeter in um.

DA — Density Area property. This is the area of the data capture window. Maximum area is the Density WINDOW value squared.

DA fill — Area of layer “fill” in the data capture window.

DA keepout — Area of layer “keepout” in the data capture window.

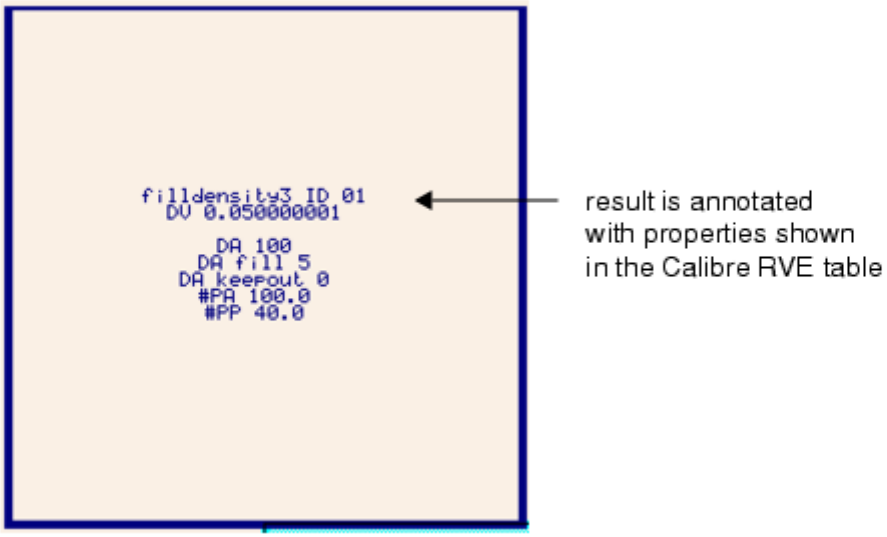
DV — Value of the Density operation expression, which is compared to the operation’s constraint.

Coordinates — x and y coordinates of the vertices.

Vertices — Vertex count.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

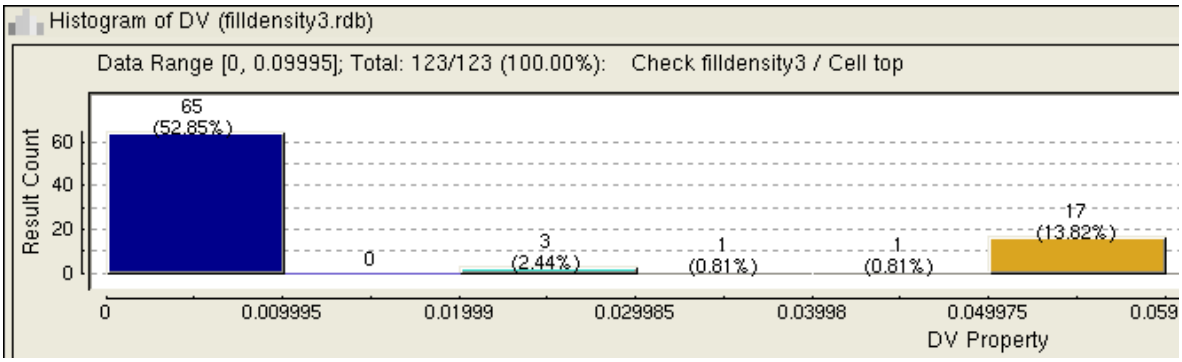
10. Select a result from the table and press **H** on your keyboard. Adjust the visibility of layers as necessary so you can see the properties associated with the result.



The annotations appear because of the selections made in Step 6.

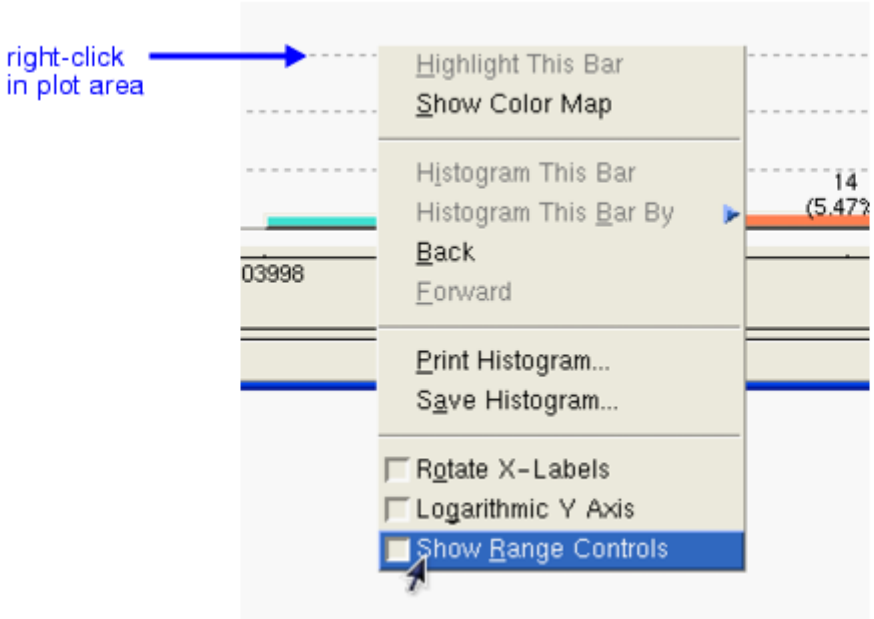
11. Right-click the **DV** column header in the Calibre RVE results table and select **Histogram DV**. This opens a histogram of the density operation values that meet the operation's constraint.

    Typically, the closer to 0 the DV value becomes, the more severe the problem.

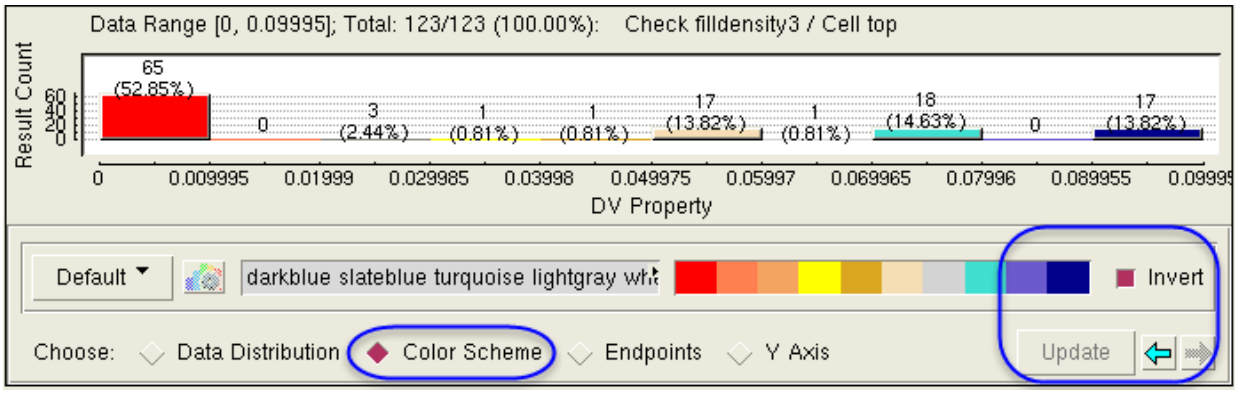


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

12. Right-click in the histogram plot area and select **Show Range Controls**.



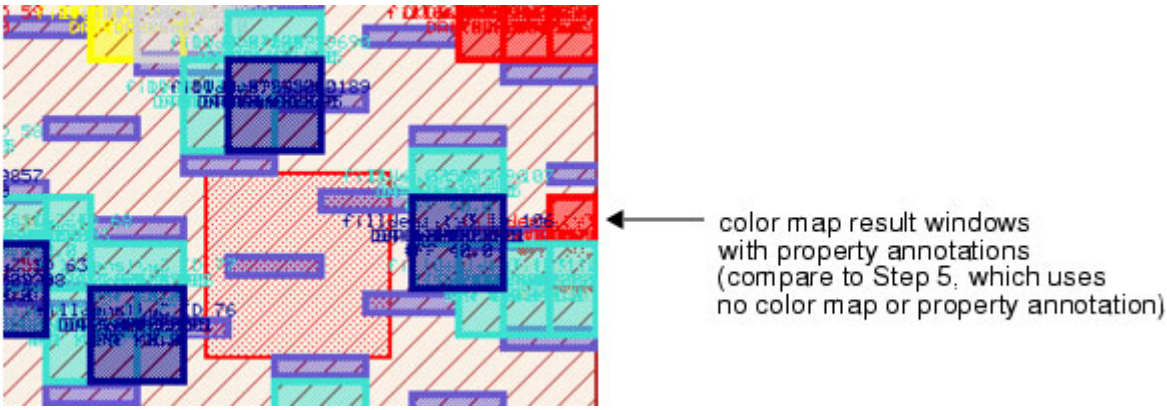
The control panel appears at the bottom of the histogram pane. These controls allow you to adjust the appearance of the histogram. In this case, the color order is inverted:



Unselect **Show Range Controls** from the right-click menu to dismiss the control panel.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

13. Right-click in the histogram plot area as in the previous step and select **Show Color Map**. The layout highlights with all density results from the RDB with colors assigned from the histogram plot.



Turning off highlighting of Check Names, Result Numbers, and Properties from Step 6 can give a clearer layout presentation when overlapping windows are displayed.

14. Double-click any histogram bar and the corresponding results highlight in your viewer with a color that matches the bar.

15. Right-click any histogram bar and a menu of options displays from which you can select.

Results

You can use these features to determine how to fix density issues in your design. You can also use the features to help you to adjust your Density layer operations to produce more meaningful results.

Related Topics

Density Operation Categories

Using Calibre RVE for DRC [Calibre RVE User's Manual]

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Density Operation Categories

This section presents typical Density checks.

Table 4-1. Density Operation Solutions

Name	Description
Finding Density Outside of a Cell or Layer	Calculate densities of a layer while avoiding a cell or another layer. This is particularly useful when you have placed IP blocks that you want to avoid when performing density checks at a higher level.
Finding Density Inside of a Layer	Calculate densities of one layer that is inside of another layer.
Finding Density Inside of a Layer and Outside of a Cell	This is a combination of the two preceding methods.
Reducing RDB Data Size with Density COMBINE	Reduce RDB data size by using the COMBINE option to merge results selectively.
Using INSIDE OF LAYER Options	Discusses the differences in the INSIDE OF LAYER options. In general, INSIDE OF LAYER should be avoided in favor of the preceding solutions in this table.

Finding Density Outside of a Cell or Layer

The following procedure demonstrates how to perform density checking outside of a cell by using an exclusion layer.

You may want to perform this sort of Density operation in cases like these:

- You have an IP block in your design, and it may be desirable not to perform density checking inside the placed block. Frequently, this is because the density specification for the block is different from the rest of the design.
- You perform density checking within some cell and it is clean by itself. However, after placing the cell, density checking at a higher level may no longer be clean because of the placement location. Therefore, you do not want to check the placed cell to avoid false errors.
- You may have a marker layer that is used to identify an exclusion area where density checking should not occur. This could apply in the case of a cell you want to exclude, as in the first two cases, or simply in some area of the design that is not cell-specific.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

One advantage to writing the Density operations in the way shown in this section is you can have multiple Density operations using differing constraints (but otherwise using the same parameter values) and they all run concurrently.

Prerequisites

- Thorough understanding of the Density AREA() function and math operators.

Procedure

1. Write the following operation to derive an exclusion layer called keepout where density is not measured:

```
keepout = EXTENT CELL "cell" ORIGINAL
```

2. Write a density check using the keepout layer from Step 1, as follows:

```
filldensity = DENSITY fill keepout < d
              [(AREA(fill)/!AREA(keepout))/AREA()]
              WINDOW w STEP s
              RDB filldensity.rdb

filldensity { COPY filldensity }
```

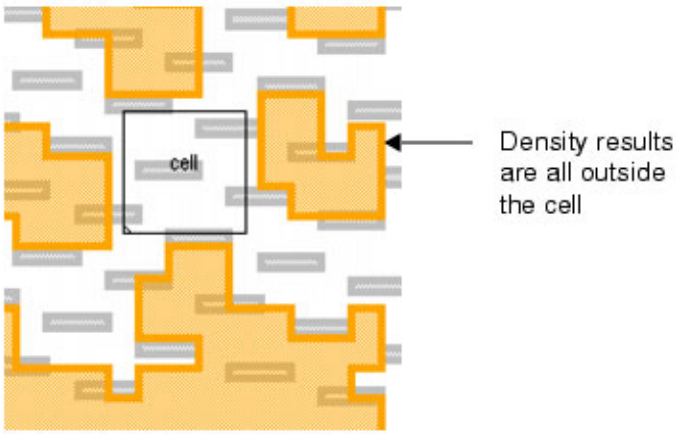
The d, w, and s parameters in this rule can be chosen for your process and set up as variables in the rule file.

The expression (AREA(fill)/!AREA(keepout))/AREA() measures the density of the fill layer in the data capture window wherever the keepout layer is *absent*. The !AREA(keepout) function acts as a detector of whether the keepout layer is present. If keepout is present, !AREA(keepout) returns a 0. Division by 0 due to the !AREA(keepout) function does not satisfy the Density operation constraint, so any windows where this function is 0 are not output.

Results

The results may appear as follows:

Figure 4-1. Density Outside of a Cell



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

Analyzing Density Results with Calibre RVE

Density Operation Categories

Finding Density Inside of a Layer

The following procedure shows Density operations for checking polygons from one layer that are inside of another layer. The operation has an INSIDE OF LAYER option, but its performance can be slow in certain situations, so it is best to avoid using that option.

One advantage to writing the Density operations in the ways shown in this section is you can have multiple Density operations using differing constraints (but otherwise using the same parameter values) and they will all run concurrently.

The d, w, and s parameters shown in this procedure can be chosen for your process and set up as variables in the rule file.

Prerequisites

- Thorough understanding of the Density AREA() function and math operators.

Procedure

1. Write a Density check as follows, where you want to check the fill layer inside the marker layer:

```
filldensity = DENSITY fill marker < d
              [AREA(fill)/AREA(marker)]
              WINDOW w STEP s
              RDB filldensity.rdb

filldensity { COPY filldensity }
```

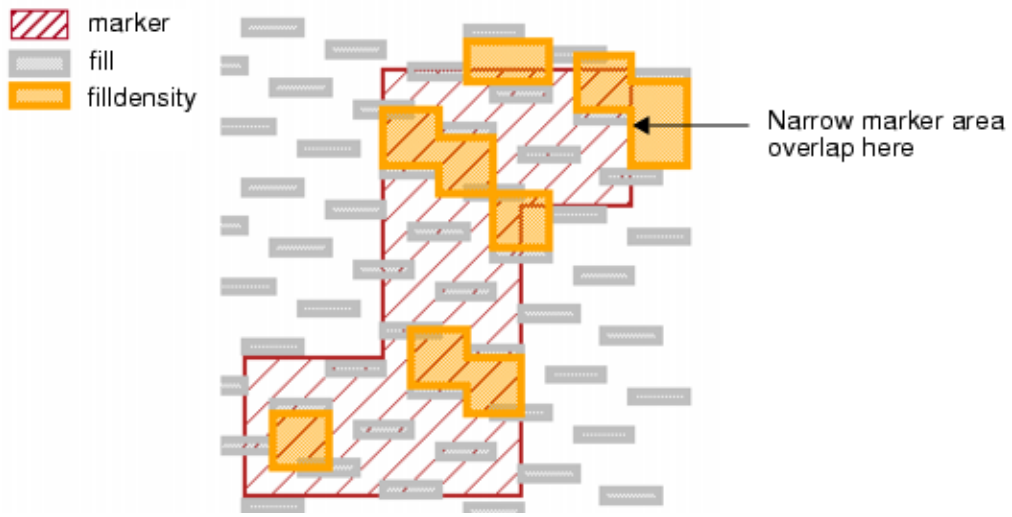
   The AREA(fill)/AREA(marker) expression calculates the quotient of the area of the fill layer versus the area of the marker layer in the data capture window. If AREA(marker) is 0, that window is not output.

   A potential drawback to this expression is the denominator may not generally produce constant values.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

2. Run DRC and view the results for the filldensity layer. Results could be as follows:

Figure 4-2. Density Inside of Marker Layer



Notice in Figure 4-2 that all output polygons have at least some interior points coincident with the marker layer. Again, a possible disadvantage to having AREA(marker) in the denominator is this function would not produce constant values.

3. A modified version of the operation from Step 1 is this:

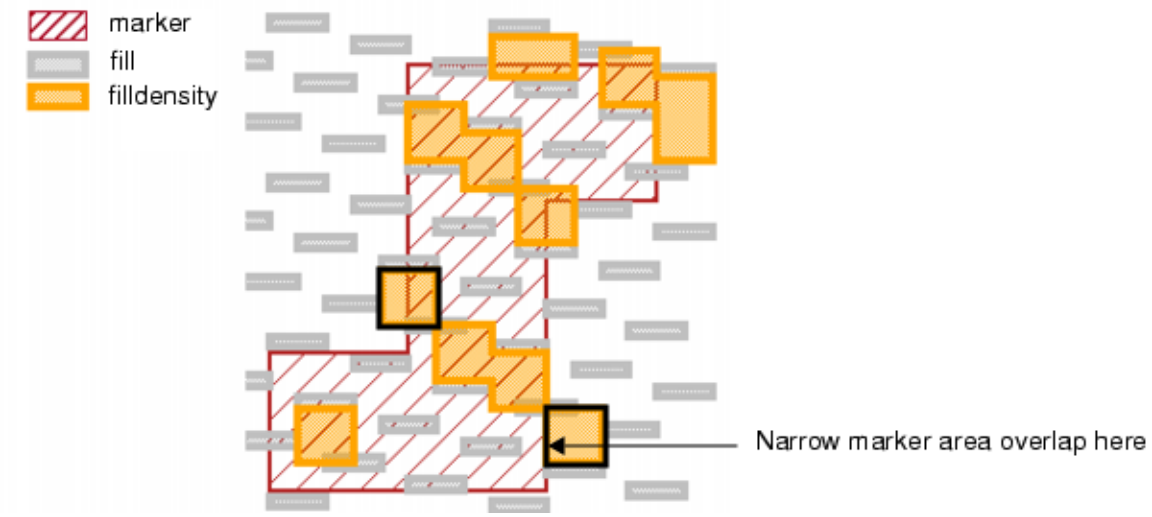
```
filldensity = DENSITY fill marker < d
              [(AREA(fill)/!!AREA(marker))/AREA()]
              WINDOW w STEP s
              RDB filldensity.rdb

filldensity { COPY filldensity }
```

The (AREA(fill)/!!AREA(marker))/AREA() expression calculates the quotient of the area of the fill layer versus the area of the data capture window, *if there is marker layer area* in the window. This form has the advantage of keeping the value of the denominator constant while guaranteeing the marker layer must be present in order to produce a result.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

4. Run DRC and view the results for the filldensity layer. Results could be as follows:

Figure 4-3. Density Inside of Marker Layer (Improved)



Compare these results to Figure 4-2. There are two additional results (outlined in black) due to the use of the different expression.

A drawback to this method (and the one from Step 1) is interpreting results that occur around the perimeter of the marker layer can be tricky. Are the gaps in the filldensity regions around the marker perimeter passing or failing?

5. Another method is to ensure that a data capture window shares at least 80 percent of its area with the marker layer before that window's measurement can meet the constraint. Write this operation:

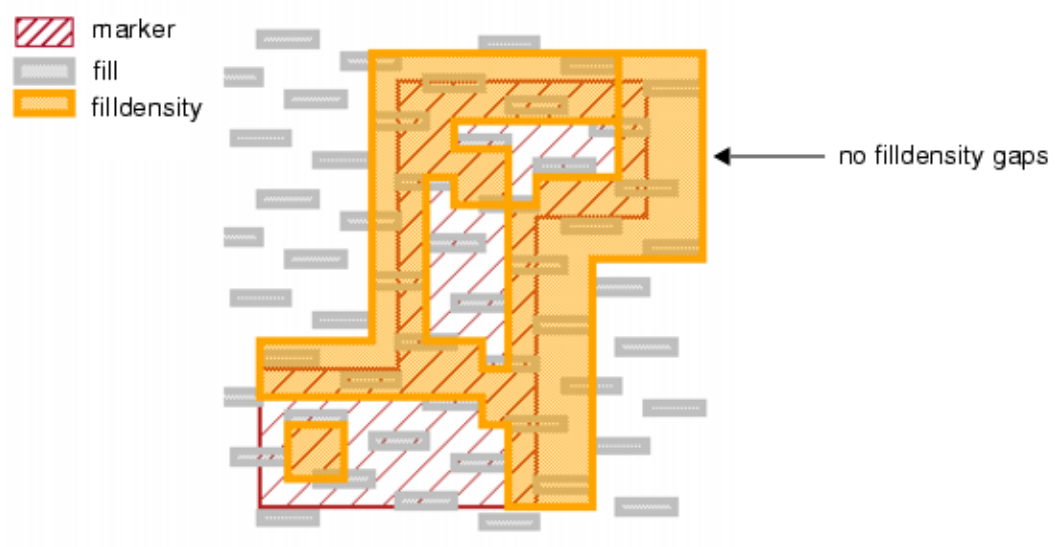
```
filldensity = DENSITY fill marker < d
    [(~~(AREA(marker)-0.8*w*w)*(AREA(fill)/!!AREA(marker)))/AREA()]
              WINDOW w STEP s
              RDB filldensity.rdb

filldensity { COPY filldensity }
```

The expression uses elements of the expression found in Step 3, and those elements perform the same function here. The ~~(AREA(marker)-0.8*w*w) portion of the expression returns a 1 if the area of the marker layer intersects at least 80 percent of the window. Otherwise this portion of the expression returns a 0, which falls within the Density constraint.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

6. Run DRC and view the results for the filldensity layer. Results could be as follows:

Figure 4-4. Density Using Minimum Marker Layer Overlap



Compare this with Figure 4-2. Unless the data capture window shares 80 percent of its area with the marker layer, the window fails.

This method makes for cleaner results presentation in certain respects because it treats the results around the perimeter of the marker layer as failing. The challenge is then to separate the results in the interior of the marker layer from those around the perimeter. This is done using Calibre RVE.

7. Open the *filldensity.rdb* file in Calibre RVE and sort the DA marker column in descending order. Scroll down until you see DA marker values that are less than 80 percent of the maximum corresponding value in the DA column.

DA marker sorted in descending order

DA	DA fill	DA marker	DV
100	5	100	0.0500000
100	7.50000	99.9000	0.0750000
100	5	99.9000	0.0500000
100	10	75.0250	0
75	0	75	0
100	10.0200	75	0

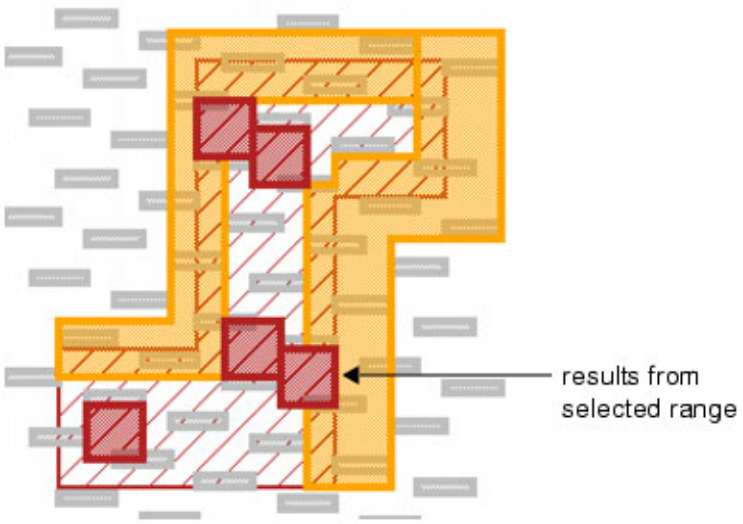
DA marker < 0.8 * DA maximum

DV is 0 (failing) from this point downward

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

8. Highlight all results where DA marker is at least 80 percent of DA. (You can Shift+click a range of rows in the Calibre RVE results table.) Results could be as follows:

Figure 4-5. Windows Inside Marker Layer



Compare this with Figure 4-4. These are the windows inside the marker layer that fail.

Results

You can adjust the Density constraint, WINDOW, and STEP values and view them using similar techniques shown in this section. Using these methods, you can optimize your results presentation.

Related Topics

Analyzing Density Results with Calibre RVE

Density Operation Categories

Finding Density Inside of a Layer and Outside of a Cell

If you want to exclude a placed cell from density checking, but you want to check inside of a boundary layer at the same time, this procedure shows an optimized method.

This procedure is a combination of the "Finding Density Outside of a Cell or Layer" and the "Finding Density Inside of a Layer" topics.

The Density operation has an INSIDE OF LAYER option which could apply in this case, but its performance can be slow in certain situations, so it is best to avoid using that option.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

One advantage to writing the Density operations in the way shown in this section is you can have multiple Density operations using differing constraints (but otherwise using the same parameter values) and they will all run concurrently.

Prerequisites

- Thorough understanding of the Density AREA() function and math operators.

Procedure

1. Write the following operation to derive an exclusion layer where density is not performed:

```
keepout = EXTENT CELL "my_cell" ORIGINAL
```

2. Write a Density check using the derived layer from Step 1. You can write a check of the fill layer inside the boundary layer, but outside the keepout layer, as follows:

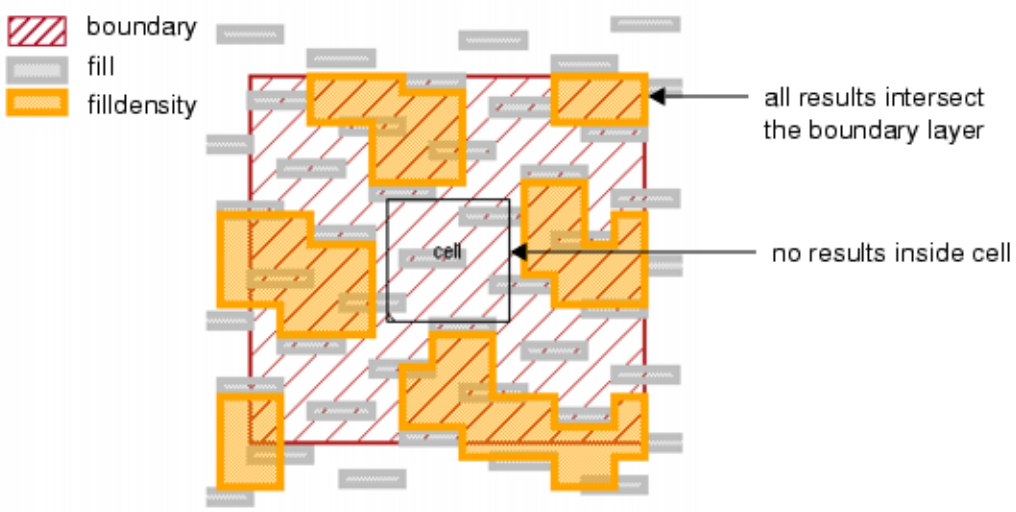
```
filldensity = DENSITY fill boundary keepout < d
    [(AREA(fill)/(!!AREA(boundary)*!AREA(keepout)))/AREA()]
              WINDOW w STEP s
              RDB filldensity.rdb

filldensity { COPY filldensity }
```

   The d, w, and s parameters in this rule can be chosen for your process and set up as variables in the rule file. These are interpretations of portions of the expression

   The (AREA(fill)/(!!AREA(boundary)*!AREA(keepout))) portion of the expression returns the area of the fill layer inside the data capture window when the boundary layer is present but the keepout layer is not. Division by 0 does not satisfy the constraint, so any case where this occurs in the sub-expression is not output.

3. Run DRC and view the results for the filldensity layer. Results could be as follows:

Figure 4-6. Density Inside of Boundary Layer and Outside a Cell



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

4. Open the *filldensity.rdb* file in Calibre RVE and sort the DA boundary column in descending order. Scroll down until you see DA boundary values that are less than 99 percent of the maximum DA value.

DA boundary sorted in descending order

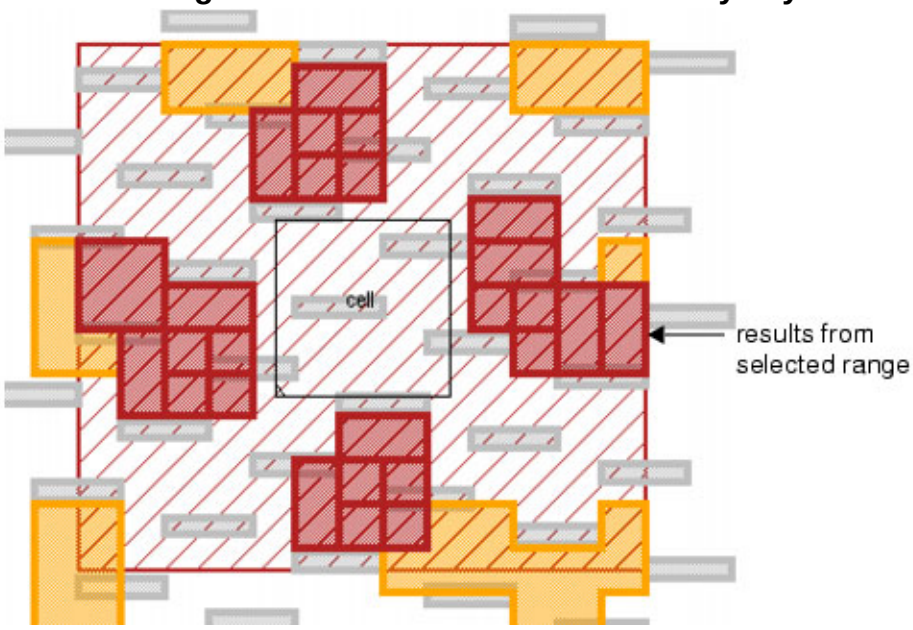
DA	DA boundary	DA fill	DA keepout	DV
100	100	5	0	0.0500000
100	100	5	0	0.0500000
100	99.9000	7.50000	0	0.0750000
100	99.9000	5	0	0.0500000
75	75 < 0.99 * max DA	0	0	0
75	75	0.0200000	0	0.000266667

Results from this point upward are completely inside the marker layer

The 99 percent cutoff point is arbitrary, but is useful when you want to show only the result windows that are inside the boundary layer.

5. Highlight all results where DA boundary is at least 99 percent of the maximum DA value. (You can Shift+click a range of rows in the Calibre RVE results table.) Results could be as follows:

Figure 4-7. Windows Inside Boundary Layer



Compare this with Figure 4-6. These are the windows inside the boundary layer that fail.

Results

You can adjust the Density constraint, WINDOW, and STEP values and view them using similar techniques shown in this section. Using these methods, you can optimize your results presentation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

Analyzing Density Results with Calibre RVE

Density Operation Categories

Reducing RDB Data Size with Density COMBINE

This procedure shows how to use the Density COMBINE option together with Calibre RVE.

Density results sent to the DRC Results Database are in merged form. This sacrifices results detail in favor of reduced data size and simplicity of results presentation. RDB results are not merged; hence, greater detail is available at the expense of increased data size and complexity of results presentation.

In certain scenarios, such as with IP blocks in the design, the block's placement location can result in false density errors because the Density WINDOW grid no longer aligns with the base point of the block as when the block is unplaced. To mitigate these errors, small STEP sizes are used, which dramatically increases the number of RDB results (and the file size).

The Density COMBINE option allows you to consolidate RDB result windows into more manageable merged regions. The COMBINE option also provides minimum, average, and maximum density values for a given region, which is helpful in interpreting the results.

The d, w, and s parameters shown in this procedure can be chosen for your process and set up as variables in the rule file.

Prerequisites

- Thorough understanding of the Density AREA() function and math operators.

Procedure

1. Add the COMBINE >= 0 option to an existing Density check, as follows:

```
m1_density {
@ Density m1 < ^d% outside of keepout layer
  DENSITY m1 keepout < d
  [(AREA(m1)/(!AREA(keepout)))/AREA()] WINDOW w STEP s
  COMBINE >= 0
  RDB density_m1.rdb
}
```

   This checks the density of m1 outside of a keepout layer. The COMBINE >= 0 option causes RDB results windows to be combined (merged) if they share edges.

2. Open the *density_m1.rdb* file in Calibre RVE.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 4-8. Combined Density RDB Results

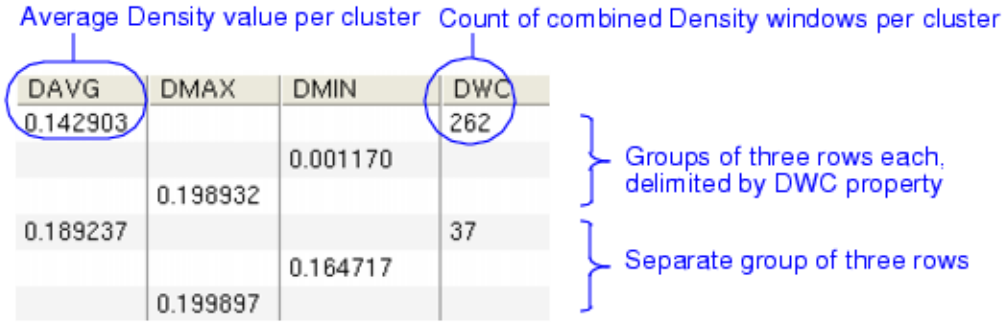


Figure 4-8 shows a typical table of results. Property definitions are these:

DAVG — Average density value for all windows in a cluster.

DMAX — Maximum density value for one window in the cluster.

DMIN — Minimum density value for one window in the cluster.

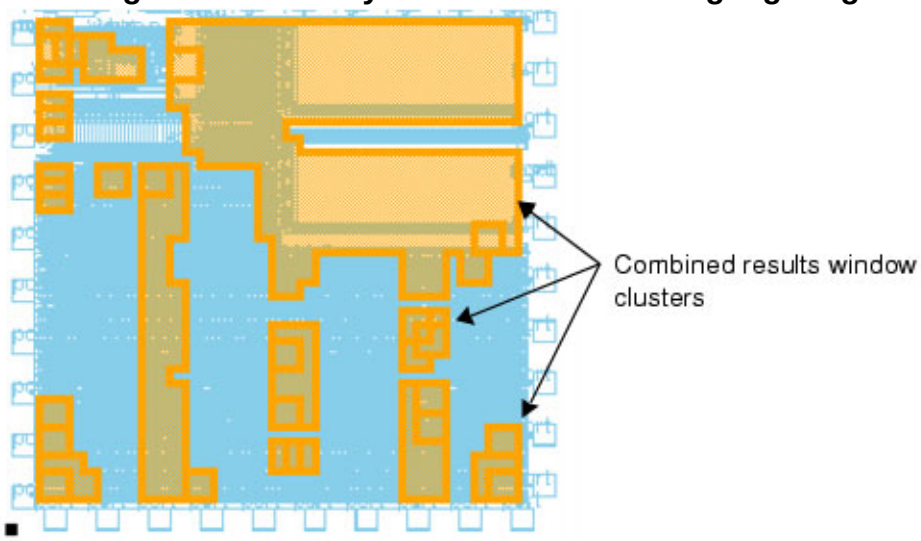
DWC — Number of windows that have been combined into a cluster.

Coordinates (not shown) — Vertex coordinates.

Vertices (not shown) — Vertex count.

3. Highlight all RDB results in your viewer by selecting the top-level cell in the results tree and pressing **H** on your keyboard. The results organized by cluster are shown, such as in Figure 4-9.

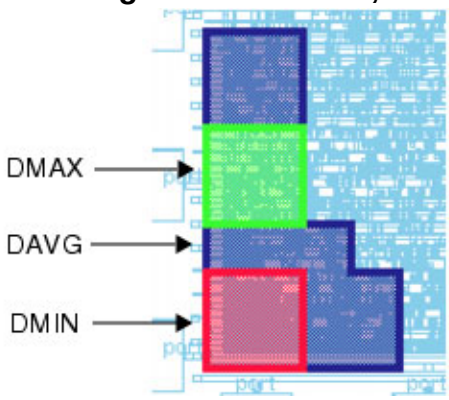
Figure 4-9. Density COMBINE Results Highlighting



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

4. Individually highlight the DAVG, DMAX, and DMIN rows from a group, as shown in Figure 4-8. The results could appear as follows:

Figure 4-10. DAVG, DMIN, and DMAX Row Highlights



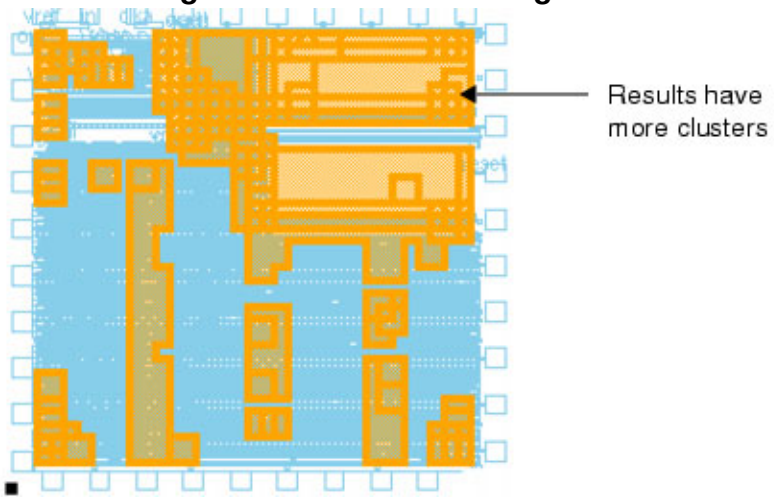
5. Modify the rule check from Step 1 by specifying a different COMBINE constraint as follows:

```
m1_density {
@ Density m1 < ^d% outside of keepout layer
  DENSITY m1 keepout < d
  [(AREA(m1)/(!AREA(keepout)))/AREA()] WINDOW w STEP s
  COMBINE <= 0.20 // combine windows with <= 20 percent
                  // relative difference in density value
  RDB density_m1.rdb
}
```

Now, instead of combining any adjacent results windows, the windows that have at most 20 percent relative difference in density values are combined.

6. Open the *density_m1.rdb* file in Calibre RVE and highlight all results in the top cell. The results may appear as follows:

Figure 4-11. Results Using COMBINE <= 20



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Compare Figure 4-11 with Figure 4-9. While the highlighted areas are the same, there are more results clusters in this case because of how the windows have been combined. The COMBINE <= 0.20 setting combined fewer results windows into each cluster, so there are more clusters to display.

You can highlight individual rows from a group of three as in Step 4.

Results

The default mode for the COMBINE is the RELATIVE mode, where the percentage difference in density between adjacent windows is used to determine if results should be combined. There is also an ABSOLUTE mode, where the simple difference in density between adjacent windows is used. See the Density COMBINE option description in the *SVRF Manual* for details.

Related Topics

Analyzing Density Results with Calibre RVE

Density Operation Categories

COMBINE Output [Standard Verification Rule Format (SVRF) Manual]

Using INSIDE OF LAYER Options

In general, the use of Density INSIDE OF LAYER option is discouraged for performance reasons. Using the other methods shown in this chapter is preferred for most situations. However, if you choose to use INSIDE OF LAYER, this section discusses the main variants of this option.

The d, w, and s parameters shown in this procedure can be chosen for your process and set up as variables in the rule file.

Procedure

1. Write a Density check using the INSIDE OF LAYER option, as follows:

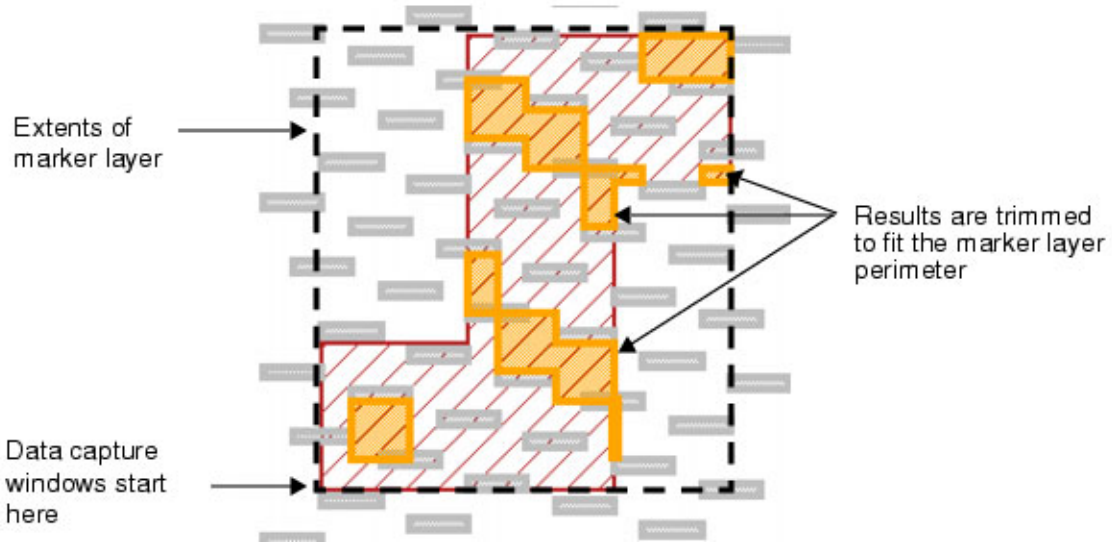
```
filldensity = DENSITY fill < d WINDOW w STEP s
              INSIDE OF LAYER marker
              RDB filldensity.rdb

filldensity {
@ Density fill < ^d INSIDE OF LAYER marker
COPY filldensity
}
```

This finds the density of the fill layer inside of the marker layer.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

2. Highlight the results of the filldensity rule check in Calibre RVE. The results might appear as follows:

Figure 4-12. Density INSIDE OF LAYER marker



The Density window grid starts at the basepoint of the extents of the marker layer polygons. Results are just those regions that intersect the marker layer.

3. Change the rule check from Step 1 to include the CENTERED option, as follows:

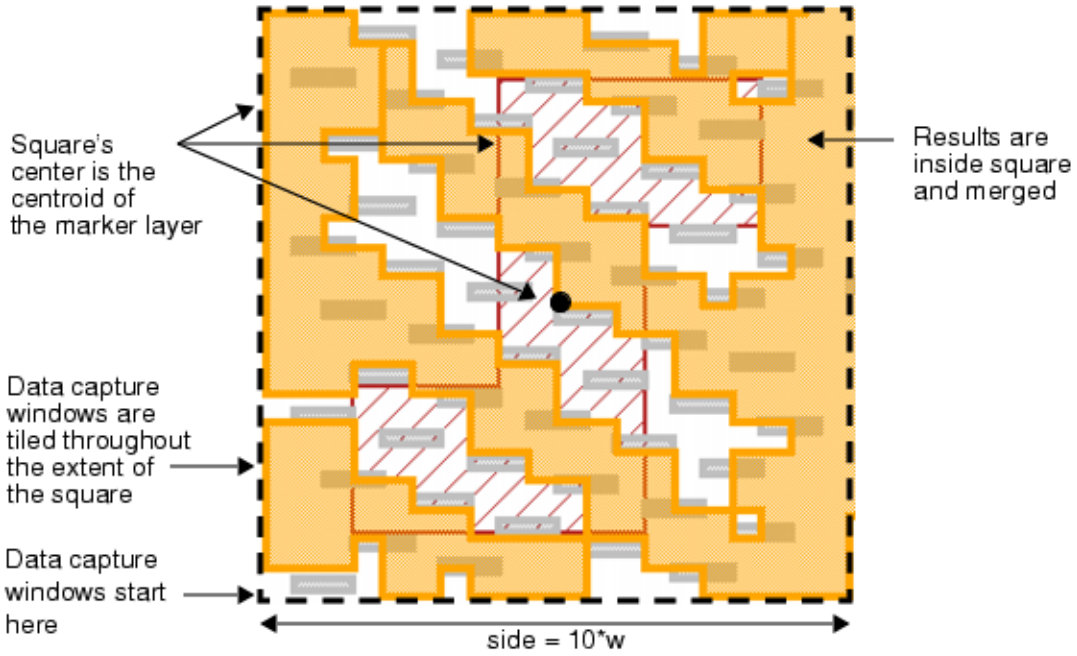
```
filldensity = DENSITY fill < d WINDOW w STEP s
              INSIDE OF LAYER marker CENTERED 10*w
              RDB filldensity.rdb

filldensity {
@ Density fill< ^d INSIDE OF LAYER marker CENTERED 10*w
COPY filldensity
}
```

The CENTERED option determines the centroid of each marker layer polygon. Then a square is constructed of side length equal to the CENTERED argument (in this case, 10 times the window width). The center of the square is the centroid of the marker layer polygon. The square is then used as the region for tiling the data capture windows.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

4. Highlight the results of the filldensity rule check in Calibre RVE. The results might appear as follows:

Figure 4-13. Density INSIDE OF LAYER marker CENTERED



Compare Figure 4-13 and Figure 4-12.

5. Change the rule check from Step 1 to include the BY POLYGON option, as follows:

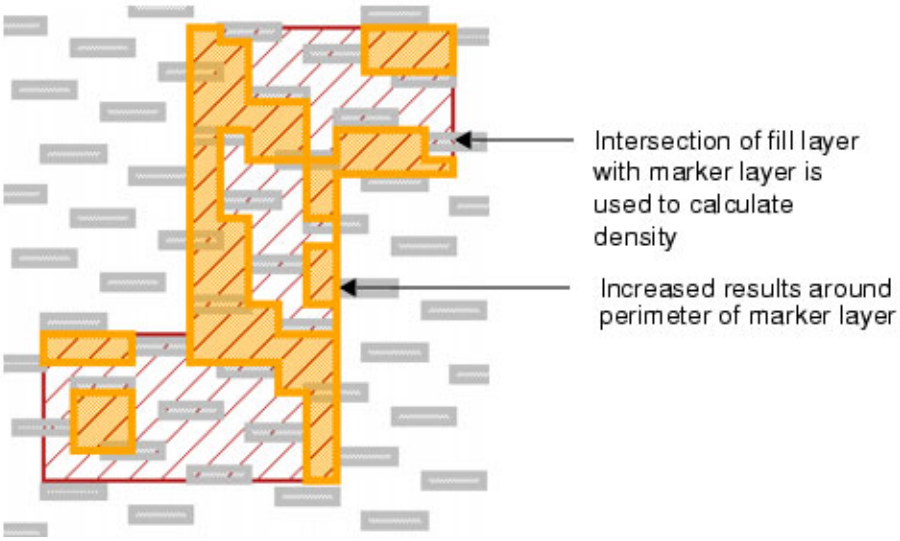
```
filldensity = DENSITY fill < d WINDOW w STEP s
              INSIDE OF LAYER marker BY POLYGON
              RDB filldensity.rdb

filldensity {
@ Density fill< ^d INSIDE OF LAYER marker BY POLYGON
COPY filldensity
}
```

The BY POLYGON option causes the intersection of the fill and marker layers to be taken first, and then the INSIDE OF LAYER option proceeds as in Figure 4-12.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

6. Highlight the results of the filldensity rule check in Calibre RVE. The results might appear as follows:

Figure 4-14. Density INSIDE OF LAYER marker BY POLYGON



The main effect of BY POLYGON is to increase results around the perimeter of the marker layer in comparison to INSIDE OF LAYER by itself. Compare Figure 4-14 with Figure 4-12.

7. Change the rule check from Step 5 to use the BY RECTANGLE option, as follows:

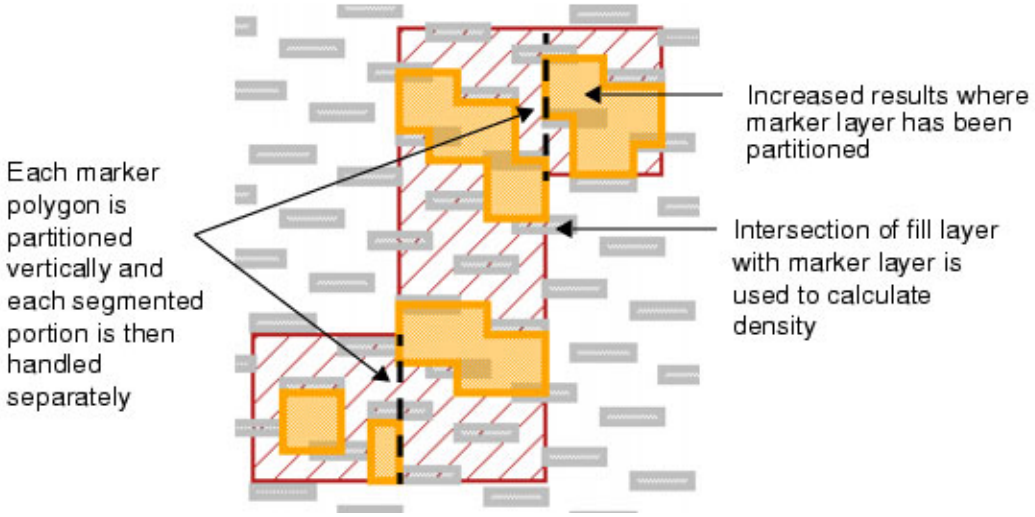
```
filldensity = DENSITY fill < d WINDOW w STEP s
              INSIDE OF LAYER marker BY RECTANGLE
              RDB filldensity.rdb

filldensity {
@ Density fill< ^d INSIDE OF LAYER marker BY RECTANGLE
COPY filldensity
}
```

The BY RECTANGLE option first vertically partitions the marker layer polygons into trapezoids and triangles. Then, for each trapezoid or triangle, the BY POLYGON behavior proceeds as in Figure 4-14.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

8. Highlight the results of the filldensity rule check in Calibre RVE. The results might appear as follows:

Figure 4-15. Density INSIDE OF LAYER marker BY RECTANGLE



BY RECTANGLE causes each segmented portion of the original marker polygon to be treated individually as a separate marker polygon. Then the BY POLYGON algorithm is used. Compare Figure 4-15 and Figure 4-14 to see the differences between BY RECTANGLE and BY POLYGON.

Results

You can adjust the Density constraint, WINDOW, and STEP values to optimize your results presentation.

Related Topics

Analyzing Density Results with Calibre RVE

Density Operation Categories

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Chapter 5
Empty Layers

This chapter discusses management of empty layers including how to assign them, how to find them, and how to ignore them.

Specification of an Empty Layer 135
Derivation of Empty Layers 135
Empty Layer Checks 136

Specification of an Empty Layer

An empty layer can be specified in the rule file.

The following example shows how to specify a layer that is empty:

```
LAYER empty 999 // no data on this layer
LAYER MAP 999 DATATYPE >=0 1000 // all polygon data on 999 goes to 1000
LAYER MAP 999 TEXTTYPE >=0 1000 // all text on 999 goes to 1000
LAYER from_999 1000 // everything from 999 is mapped here.
```

The layer 999 contains no data and may be used for suitable purposes.

Related Topics

Derivation of Empty Layers

Empty Layer Checks

Derivation of Empty Layers

Empty layers can be derived in the rules. The following examples show the three possible derived layer types.

```
// empty polygon layer
empty_shape = AREA(EXTENT) == 0

// empty edge layer
empty_edge = LENGTH(EXTENT) == 0

// empty error cluster layer
empty_error = EXT empty_shape < 1
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

These layer derivations can be used inside rule checks if desired; however, empty_error can only be passed to a layer operation that accepts error layers as input. An empty output layer is permitted in the DRC Results Database or in a DRC Check Map database.

Empty Layer Checks

In certain situations, it is desirable to check if a particular layer is empty.

In a TVF FUNCTION, you can use the following:

```
if {[tvf::IS_LAYER_EMPTY $layer]}
  # layer is empty
  ...
} else {
  # layer is non-empty
  ...
}
```

The tvf::IS_LAYER_EMPTY command returns a 1 if the layer is empty and a 0 otherwise. This command has excellent performance and is probably the best solution in most cases. It could be used as follows:

```
TVF FUNCTION empty_layer [/*
    tvf::GET_LAYER_ARGS inputLayer myExtent
    if {[tvf::IS_LAYER_EMPTY $inputLayer]} {
         tvf::OUTLAYER AREA $myExtent == 0
    } else {
         tvf::OUTLAYER COPY $myExtent
    }
*/]

extent_ = EXTENT DRAWN ORIGINAL
// if in_layer is empty, empty1 is empty. otherwise, empty1 is the extent
// of the database.
empty1 = TVF empty_layer in_layer extent_
```

The following uses traditional SVRF coding:

```
// x is the layer whose emptiness is of interest
chip = EXTENT DRAWN ORIGINAL
chip_x = chip INTERACT x

M1.space {
// if x is not present, measure this
EXT (M1 NOT chip_x) < 0.5
// else measure this
EXT (M1 AND chip_x) < 0.3
}
```

The motivation here is to derive a layer covering the extent of a layer of interest (x in this case). Then, if that layer is present, derive a certain result; otherwise, derive another result. The EXT operation is arbitrary here. The NOT and AND operations are also arbitrary to a certain degree;

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

yet the typical implementation is to use inverse layer operations to cover the cases of whether chip_x is present or not.

For global layer checking related to (possibly) empty layers, the following table shows some useful options.

Table 5-1. Global Empty Layer Checks

Rule File Statement	Description
Layout Input Exception Severity EMPTY_LAYER 1	Lists the empty simple layers in the transcript. This is the default setting.
Layout Input Exception Severity UNUSED_DATA 2	Issues a warning in the transcript for each layer and datatype pair that are not used in the run. The default is 1, which means to list all layers and datatypes in the transcript.
Layout Input Exception Severity DATA_MAP_SOURCE 1	Issues a warning in the transcript when a simple layer required for the run has only some of its data transferred to other layers by Layer Map statements. This is the default setting.
Layout Input Exception Severity DATA_MAP_TARGET 1	Issues a warning in the transcript when a simple layer required for the run has additional data transferred to it by a Layer Map statement. This is the default setting.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Chapter 6
Layout Magnification

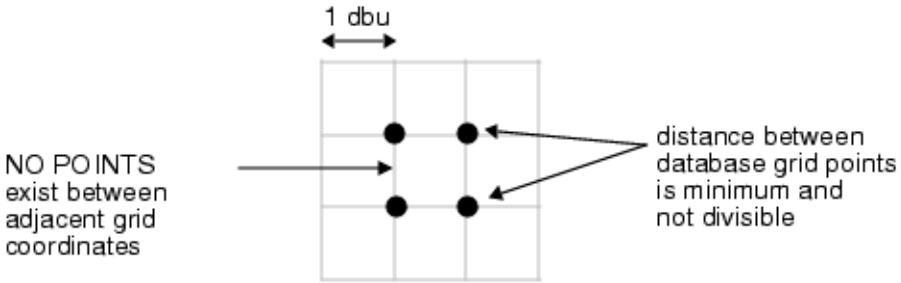
Layout magnification occurs in two ways: scaling the data by adjusting precision values (that is, changing the distance represented by a database unit) and using magnification factors to expand or shrink layout data. These are related effects and have aggregate outcomes.

Considerations When Magnifying Data 139
Layout Database Versus Calibre Database 140
Key Specification Statements for Magnification 141
Magnification and Precision Management 144
Increasing Measurement Precision 147
Reducing Layout Magnification 149
Degenerate Boundary and Polygon Management 150

Considerations When Magnifying Data

Layout data uses a *discrete coordinate system*. There is a smallest unit of distance in a layout (called a database unit or *dbu*) that is indivisible. In other words, there is no infinitesimal approach (as in the real number system) to a given point in a layout.

Figure 6-1. Discrete Database Coordinates



All distances are a multiple of that database unit. For example, if the database unit is 1 nanometer, then the smallest measured distance in the coordinate system is 1 nm. There is no such thing as a 0.5 nm distance between vertices when the database grid unit is 1 nm.

This has important implications for layout magnification, because vertices must be snapped to the physical grid after magnification is performed. Magnification is also performed hierarchically, so cell placements themselves may snap and are also affected by any grid snapping that has occurred within cells they reference. Hence, polygon vertices and cell instances can end up in unexpected locations.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Related Topics

Layout Magnification

Magnification and Precision Management

Layout Database Versus Calibre Database

At runtime, the layout database is converted to a Calibre database in memory. Calibre loads the layout database and keeps the number of grid points between two vertices the same. However, the distance between grid points may or may not be the same as the layout. If the distances are different, then scaling of the data occurs in the Calibre database. Additionally, magnification record values in the layout, as well as magnification factors specified in the rule file, are applied to the Calibre database.

The preceding considerations all apply as an aggregate magnification of the layout data within the Calibre database. Magnification changes the number of grid points between two points in the layout database.

Therefore, these things should always be borne in mind when magnification occurs:

- Pay attention to what the output grid unit is in comparison to the input grid unit.
- When decreasing layout magnification, consider increasing the rule file precision and the output database precision. If you do this, grid snapping of cell placements and layers can occur. To mitigate this, you can use Size … UNDEROVER with a 1 measurement dbu value on the affected layers to fill in possible gaps due to grid snapping. Be aware that Size can cause longer runtimes due to layer merging across the hierarchy. Another method is to adjust the measurement constraints for affected layers by 2 dbu measurement values.
- Any output layout data should be validated after magnification before combining with other layout data.
- All rule file elements that manage magnification should be validated before combining with other rule processes.

Related Topics

Key Specification Statements for Magnification

Magnification and Precision Management

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Key Specification Statements for Magnification

Certain rule file specification statements govern how magnification is performed for input or output.

Before summarizing the specification statements, it is important to understand the following two terms:

- **Physical precision** — This is the minimum fraction of a user unit represented in the layout database. Typically, the user unit is the micron (1E-06 meters) and the physical precision is 0.001 um. Hence, 1/1000 of a micron (or 1 nanometer) is the minimum fraction of a user unit represented in the layout and is the database unit. In the rule file, the Layout Precision statement (or the Layout Path PREC keyword) controls checking of the physical precision at runtime.
- **Rule file (measurement) precision** — This is the number of database units per user unit. In Calibre, this is the reciprocal of the minimum fraction of a user unit for which measurements can be taken by layer operations in the rule file. For example, a precision specification of 1000 means measurements can be taken to the nearest 1/1000 of a user unit. In most cases, this means to the nearest 0.001 micron, or 1 nanometer. This is set with the Precision statement.

If the rule file precision and the physical precision agree, then the units are in a ratio of 1:1 and no scaling occurs. Suppose the rule file Precision is 10000 and the physical precision is 0.001. This implies the measured units versus physical units are in the ratio of 0.0001:0.001 or 1:10. In this case, the data is scaled by a factor of 1/10 (or shrunk by a factor of 10). Magnification effects such as these have to be taken into account in your flow.

The following table is a summary of statements that affect magnification.

Table 6-1. Specification Statements Affecting Magnification

Statement	Description
Input statements	
Precision	Specifies the number of database units in a user unit. Default setting is 1000, which corresponds to a measurement precision of 0.001 user units. By default, having a rule file Precision that does not agree with the physical precision of the layout results in a fatal error. This can be overridden by various rule file settings.
Layout Precision	Specifies the reciprocal of the physical precision of the layout. This is used for verifying the layout precision only; it has no impact on measurements or output precision. Specifying this statement allows the rule file Precision to differ from the physical precision without causing an error. The same function is performed by the Layout Path PREC secondary keyword.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 6-1. Specification Statements Affecting Magnification (cont.)

Statement	Description
Layout Use Database Precision	Specifies that the physical precision of the layout be used for the rule file Precision value.
Layout Magnify	Specifies to multiply all coordinate values in the input database by a specified value. Used to expand or shrink input data. The same function is performed by the Layout Path MAG secondary keyword.
Layout Input Exception Severity	Specifies how certain exception conditions are handled (ignored, warning, fatal error, and so forth). The exceptions of special note in the context of magnification are these: METRIC_INPUT_FILE — handles database unit consistency for multiple GDSII layout input files. METRIC_RULE_FILE — handles agreement of the physical precision with the rule file Unit Length. PRECISION_INPUT_FILE — handles physical precision consistency for multiple layout input files. PRECISION_LAYOUT — handles agreement of the physical precision with the Layout Precision specified in the rule file. PRECISION_RULE_FILE — handles agreement of the physical precision with the rule file Precision. POLYGON_DEGENERATE — handles polygons that are not properly formed, possibly due to magnification or precision issues. POLYGON_ZERO_AREA — handles the special class of improper polygons having zero area.
Output statements	
DRC Results Database Precision	Sets the physical precision of the DRC results output. The reciprocal of the output physical precision is specified in the statement. By default, this setting is consistent with the rule file Precision. If the output physical precision is less than the input physical precision, data is shrunk. Conversely, if the output physical precision is greater than the input physical precision, data is expanded.
DRC Magnify Results	Specifies to multiply all coordinate values in the Calibre database by a specified value before output to the DRC Results Database. Can be used to expand or shrink mask output data.
DRC Magnify Density	Applies a magnification to Density RDB results upon output. This functions independently from DRC Magnify Results.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 6-1. Specification Statements Affecting Magnification (cont.)

Statement	Description
DRC Magnify NAR	Applies a magnification to Net Area Ratio RDB results upon output. This functions independently from DRC Magnify Results.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Magnification and Precision Management

Physical precision, rule file precision, and magnification are interrelated. This section discusses some useful formulas for computing scaling and magnification factors. It also discusses recommended practices for setting precision and magnification values.

Scaling and Precision Differences 144
Effects on Coordinates. 145
Magnification Values Less Than 1 147

Scaling and Precision Differences

Scaling is determined from the physical precision and the rule file precision.

Assuming the following definitions of terms:

- L — Layout Precision, or the reciprocal of the physical precision.
- R — Rule file Precision, or the reciprocal of the measurement precision.
- S — Scaling factor, which is the factor by which coordinate data is multiplied due to differing precisions.

The following relation holds:

$$S = L / R$$

Example

Assume the following:

```
LAYOUT PRECISION 1000 // 0.001 user units
PRECISION 10000 // 0.0001 user units
```

Then we have this scaling factor:

$$1/10 = 1000/10000$$

Data is scaled by a factor of 1/10 upon read-in. Similar reasoning applies on output when comparing the Layout Precision to the DRC Results Database Precision.

We now introduce these terms:

- M — Layout Magnify statement multiplication factor.
- F — Final multiplication factor applied to the layout. It is the product of the scaling factor S and the Layout Magnify factor M.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following relation holds:

F = M * S

Assume the following:

```
LAYOUT PRECISION 1000 // 0.001 user units
PRECISION 10000 // 0.0001 user units
LAYOUT MAGNIFY 10
```

then this follows:

S = 1/10

M = 10

F = 1/10 * 10 = 1

Scaling and magnification have important implications for where data is placed during a Calibre run. In particular, this affects where skew edges get placed, because they are subject to their endpoints being snapped to the new grid.

Effects on Coordinates

When data is read into Calibre, layout database coordinates (dbu) are divided by the Precision statement value, and the resulting coordinate values are used in the Calibre database. If the Precision value and the Layout Precision value (reciprocal of the physical precision) are different, then scaling occurs.

When Layout Magnify is used, the coordinates are multiplied by the magnification factor. After all magnification effects are taken into account, the coordinates used in the Calibre database are rounded and truncated.

Assume the simple case where the reciprocal of the physical precision is the same as the rule file Precision and any magnification record within the layout database is 1.

For example, assume the following in the rule file:

```
PRECISION 1000      // no scaling
LAYOUT PRECISION 1000
LAYOUT MAGNIFY 3    // M = 3
```

Assume the following layout coordinates:

```
(10345, 1412409) dbu
```

After magnification (multiplication by 3 in this case) there is this:

```
(31035,4237227) dbu
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

which is represented in the Calibre database.

Now, convert these coordinates back to the original magnification. Assume the rule file contains the following:

```
DRC RESULTS DATABASE PRECISION 1000
DRC MAGNIFY RESULTS 0.333
```

After magnification (multiplication by 0.333 in this case) truncation and rounding, the output coordinates are these:

```
(10335, 1410997) dbu
```

This implies grid coordinates can differ after application of certain magnification values. This can lead to undesirable outcomes.

A better choice for DRC Magnify Results in this case would be 0.33333333. The maximum number of significant figures for the coordinates is seven, so at least seven-place accuracy is needed to restore the original coordinates. In this case, the output coordinates are these after rounding and truncation:

```
(10345, 1412409)
```

which are the original values.

This line in the run transcript can assist in telling you how many significant figures you need for magnification values:

```
:--- DATABASE EXTENT = [ 0 , 0 ] -> [ x , y ]
```

The x and y values are given in user units, from which you can deduce the number of significant figures needed. In general, choosing at least one significant figure more than the maximum number of significant figures in the database extent is a good practice.

In the more complicated case where the physical precision and the rule file precision differ, and where there is a magnification record in the database for a given cell, then the final coordinates are multiplied by all these factors. Let C be the magnification record value in the layout, let S be the scaling factor as defined previously, and let M be the Layout Magnify value. Then this is what occurs for all coordinates:

Before read-in:	(x, y) user units (floating point)
After read-in:	(C*S*M*x, C*S*M*y) database units (integer, truncated and rounded)

Similar implications for grid snapping apply as discussed previously.

Related Topics

Scaling and Precision Differences

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Degenerate Boundary and Polygon Management

Magnification Values Less Than 1

When generating mask data output and using magnification values less than 1, it is recommended to increase your measurement precision.

For example, consider these coordinates in user units:

```
(1.996, 1.654)
```

Assume the following are in the rule file:

```
PRECISION 1000
LAYOUT PRECISION 1000
LAYOUT MAGNIFY 0.9
```

The product of the layout magnification with the coordinates is this:

```
(1.7964, 1.4886)
```

Because the rule file Precision is 1000, these values are rounded to the third decimal place and the vertex coordinates are snapped to these values:

```
(1.796, 1.489)
```

In one case the coordinate is rounded down, in the other it is rounded up. In either case, if this occurs at a cell boundary, gaps in geometry can result in placements of that cell after magnification.

Referring to the discussion of the final magnification factor F under "Scaling and Precision Differences" on page 144, you can eliminate the effects of rounding by choosing these rule file values:

```
PRECISION 10000 // increased by a factor of 10
LAYOUT PRECISION 1000
LAYOUT MAGNIFY 9 PLACE
```

You still have a final magnification factor of 0.9, but because of the increase in Precision to the fourth decimal place, you eliminate the rounding and grid snapping. The PLACE keyword for Layout Magnify is recommended when reducing magnification.

Increasing Measurement Precision

Sometimes it is desirable to increase the measurement precision beyond the physical precision of the layout. This is used for DRC runs where a smaller measurement grid is desired than what the layout has inherently.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Prerequisites

- A physical layout and DRC rule file.
- Familiarity with the concepts discussed under "Considerations When Magnifying Data," "Key Specification Statements for Magnification," and "Magnification and Precision Management."

Procedure

Note

This procedure makes some assumptions about the physical precision and measurement precision. You can adjust the values in your rule file to be consistent with your own design.

1. Place the following entries in your rule file to govern layout input:

```
LAYOUT PRECISION 1000 // database unit is 1 nm.
PRECISION        4000 // measure to the nearest 0.25 nm;
                      // data is scaled by 1/4;
LAYOUT MAGNIFY   4    // place the data at original dimensions.
```

2. The following statement is optional and governs output results. The rule file Precision is used by default.

```
DRC RESULTS DATABASE PRECISION 4000 // matches PRECISION by default.
```

3. [Optional.] Use Size … UNDEROVER with the new measurement dbu value on drawn layers to fill in possible gaps due to grid snapping:

```
layer_snap = SIZE layer BY 0.00025 UNDEROVER
```

   Be aware this can have a performance penalty if it causes much layer merging across the hierarchy. Another alternative is to adjust measurement constraints by 2 dbu for layers subject to grid snapping.

Results

Results are presented in Calibre RVE using the same grid that you measured with, namely, 0.25 nm.

This method of output is not recommended:

```
DRC RESULTS DATABASE PRECISION 1000
DRC MAGNIFY RESULTS 0.25 // compensate for the magnification by 4.
```

It can lead to grid snapping on account of the loss of precision and affects the integrity of the results. As mentioned in the Procedure section, using Size … OVERUNDER is recommended.

When both Layout Magnify and DRC Magnify Results statements are used as shown in this procedure, the product of the two values should be as close to 1 as possible. If not 1 exactly, the product should give 9s to at least one decimal place beyond the maximum number of significant figures in the coordinates of your layout database. Suppose the maximum number of significant

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

figures is 7. Suppose your Layout Magnify factor is 3. To restore the original coordinates, you should use DRC Magnify Results 0.33333333. The product gives 9s to the eighth significant figure, which gives better accuracy.

Related Topics

Effects on Coordinates

Reducing Layout Magnification

Reducing Layout Magnification

It is desirable to reduce layout magnification (shrinking the layout) without causing problems due to grid snapping. This is accomplished by increasing the precision.

Prerequisites

- Physical layout and rule file.
- Familiarity with the concepts discussed under "Considerations When Magnifying Data," "Key Specification Statements for Magnification," and "Magnification and Precision Management."

Procedure

1. Choose your magnification factor and multiply it by 10. For example, if your multiplication factor is 0.9, use 9 for the following step.
2. Add the following statement to the rule file:

   ```
   LAYOUT MAGNIFY 9 PLACE // intending to use a 0.9 shrink factor
   ```

3. Use a Precision that is 10 times the Layout Precision value:

   ```
   PRECISION 10000 // scaling layout by 1/10
   LAYOUT PRECISION 1000 // database grid unit is 0.001 user units
   ```

   This set of statements in conjunction with Step 2 causes a final layout multiplication factor of 0.9, with coordinate values maintained to the fourth decimal place.

4. [Optional.] You can use Size … UNDEROVER with the new measurement dbu value on drawn layers to fill in possible gaps due to grid snapping:

   ```
   layer_snap = SIZE layer BY 0.0001 UNDEROVER
   ```

   Be aware this can have a performance penalty if it causes much layer merging across the hierarchy. Another alternative is to adjust measurement constraints by 2 dbu for layers subject to grid snapping.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Results

The DRC Results Database Precision follows the Precision statement setting by default. So in this case, the physical precision of the output is 0.0001 user units. The scaled results are reduced by a factor of 0.9 and are on the output database unit grid.

Related Topics

Magnification Values Less Than 1

Effects on Coordinates

Degenerate Boundary and Polygon Management

Degenerate Boundary and Polygon Management

A degenerate boundary (GDS) or polygon (OASIS®[1]) is a record in a layout database that represents a malformed shape.

A degenerate polygon is indicated by the following warnings in the run transcript or DRC Summary Report:

> WARNING: Degenerate (vertex count = <n>) polygon at location (x,y) in cell <cell> on layer <n> datatype <n>.
>
> WARNING: zero-area boundary at location (x,y) in cell <cell> on layer <n> datatype <n>.

By default, such polygons are discarded by Calibre on layout database read-in. These warnings should not be ignored, and the problems should be corrected.

The cause of these warnings is bad data was generated in the layout, often in this sequence:

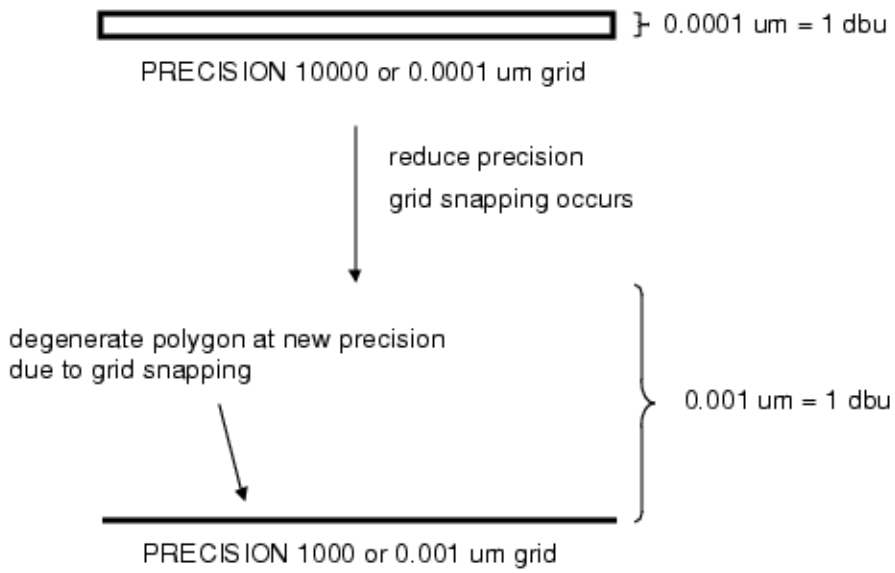
1. Polygon data was generated at a greater precision than is used for database output or than is used for the rule file Precision in a subsequent Calibre run.
2. Polygons have vertex coordinates that are on grid for the greater precision, but are not on grid at the lesser precision.
3. Polygon vertices are snapped to a new grid when the precision is reduced. As a result, they can become degenerate, and Calibre cannot process them in subsequent runs.

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.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

These types of effects are discussed under "Effects on Coordinates" on page 145. Figure 6-2 shows a simple example:

Figure 6-2. Degenerate Polygon Due to Precision Change



Avoid Degenerate Polygons on Output

You can avoid creating degenerate polygons in Calibre by checking these things:

- Ensure the rule file Precision value is less than or equal to the DRC Results Database Precision value. They are the same by default, but you should check the rule file for a DRC Results Database Precision statement.

  ```
  PRECISION 1000
  DRC RESULTS DATABASE PRECISION 10000 // OK.
  ```

  The output database precision should generally never be less than the input database precision, as this causes magnification and grid snapping issues.

  ```
  PRECISION 10000
  DRC RESULTS DATABASE PRECISION 1000 // BAD.
  ```

- Ensure that if the DRC Magnify Results statement is used that it meets one of these criteria:
  - The product of the Layout Magnify value (1 by default) and the DRC Magnify Results value is exactly 1 or gives all 9s in the decimal representation, such as 0.999999.
  - If the magnification of the output data is less than the input data, ensure that the DRC Results Database Precision value is increased by a corresponding factor. For example, if the magnification is decreased by a factor of 10 on output, then the precision should be increased by a factor of 10 to avoid grid snapping.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

See "Reducing Layout Magnification" on page 149 for a related discussion.

- Ensure that any DRC Check Map MAG and PREC keyword values are chosen using the criteria mentioned previously in this section. MAG corresponds to DRC Magnify Results and PREC corresponds to DRC Results Database Precision.

Additionally, you should use rule checks that output geometry that is subject to grid snapping in a follow-on Calibre run. Here is an example:

```
PRECISION 10000
LAYOUT MAGNIFY 10
LAYOUT PRECISION 1000
DRC RESULTS DATABASE output.gds GDS

// do this for all output layers
short_edge {@ layer edge could be snapped for PRECISION < 10000
  LENGTH layer < 0.001
}

offgrid_10 {@ layer vertex could be snapped for PRECISION = 1000
  OFFGRID layer 10
}
```

Detect Degenerate Polygons on Input

You control how degenerate polygons are handled on input using the Layout Input Exception Severity POLYGON_AREA_ZERO and POLYGON_DEGENERATE exceptions.

The default for POLYGON_AREA_ZERO is 1, which means to warn and ignore (drop) the polygon. You can set this to 2 if you want the condition to be a fatal error. The default for POLYGON_DEGENERATE is 2, which is a fatal error.

You can use Layout Input Exception RDB to store degenerate polygons when the exception settings are 0 or 1.

Related Topics

Layout Input Exception Severity [Standard Verification Rule Format (SVRF) Manual]

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Chapter 7
Layout Rotation-Sensitive Commands

This chapter identifies SVRF and Litho commands that are sensitive to rotation.

SVRF Statements Sensitive to Rotation 153
Litho Commands Sensitive to Rotation 154

SVRF Statements Sensitive to Rotation

Some SVRF statements are sensitive to rotation in general while others are sensitive to potential grid snapping induced by rotation.

The following table lists the SVRF statements that are sensitive to rotation in general:

Table 7-1. Rotation-Sensitive SVRF Statements

SVRF Statement	When used with...
Angle	
DFM Grow	
DFM Property	Measurement functions that specify X or Y directions.
DFM Space	The HORIZONTAL or VERTICAL keyword.
DFM Transform	
Enclosure	The ANGLED secondary keyword.
External	The ANGLED secondary keyword.
Grow	
Internal	The ANGLED secondary keyword.
Rotate	
Shift	
Shrink	

There is also a special case when dealing with 1 dbu long edges. For example:

```
X = EXT Y < 0.1 ABUT == 90 REGION
```

When one of the edges of Y is >= 0.1um long and the other edge is 1 grid point long, the resulting region is preserved when the long edge is horizontal. When the short edge is

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

horizontal, Calibre rounds away the error by increasing the precision. This prevents numerical precision limitations from causing erroneous output.

The following table identifies the SVRF statements that are sensitive to rotation due to potential grid snapping differences:

Table 7-2. Rotation-Sensitive SVRF Commands Due to Grid Snapping

SVRF Command
Deangle
Density
DFM Analyze
DRC Boolean Nosnap45
DRC Magnify Results… PLACE
Magnify used with angled edges.
With Width
With Neighbor

Litho Commands Sensitive to Rotation

This section lists the Litho commands that are sensitive to rotation in general. These are not used in physical verification flows but are covered here for completeness.

Table 7-3. Rotation-Sensitive Litho Commands

Litho Command	When used with...
shift (setlayer operations command)	
image (setlayer operations command)	The Shift option.
denseopc_options (setup file command)	The Shift option.
pw_condition (denseopc_options keyword)	The Shift option.
NEWTAG horizontal | vertical (nmOPC Tcl scripting command)	
horizontal_line (Sparse OPC and ORC built-in tag)	
vertical_line (Sparse OPC and ORC built-in tag)	

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Chapter 8
DRC Regression Testing

Regression testing is employed to verify tool behaviors across software versions, to benchmark rule file performance, to compare layout versions, and so forth. This testing ensures consistency of mask generation across variable factors of the design process. Constructing a robust regression flow is therefore a common concern.

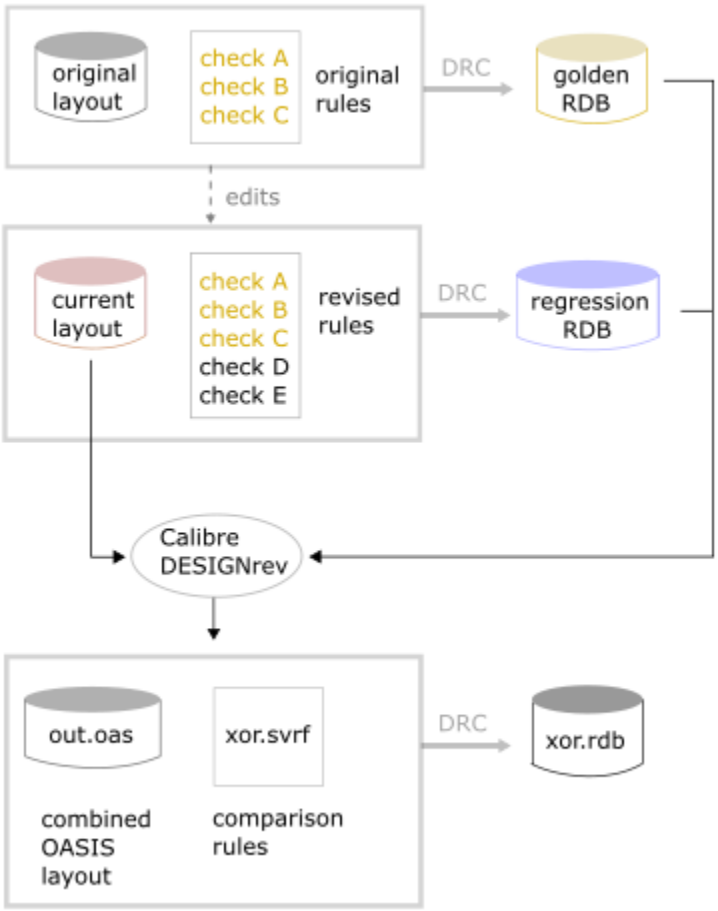
For DRC rule file regression testing, these things are assumed:

- A set of “golden” rule checks is chosen as a baseline for regression comparison.
- A “golden” DRC results database is generated using the chosen rule checks and saved for comparison against future results.
- The DRC rule file can undergo revisions except for the baseline rule check names, as these names are used for layer comparison rather than original layer names.
- Changes to rule checks (but not the names of the “golden” set of checks) can produce results that are flagged during layer comparison regression testing.
- The design layout can undergo revisions. Any changes in the geometry that correspond to the reference results are flagged during layer comparison regression testing.
- Edge layers or error cluster layers are converted and processed as polygon data to facilitate XOR comparison.

A primary goal of the rule file regression flow discussed in this chapter is to permit revision of the rule file (except baseline rule check names) while still producing useful results comparisons based upon a “golden” set of DRC results. To do this easily, the comparison must be based upon rule check names rather than layer numbers. Refer to the following figure.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 8-1. DRC Regression Flow



In order to meet the aforementioned goal, ASCII DRC results databases are used to generate an XOR comparison rule file that contains layer names based upon rule check names. This mapping of rule check names to compared layer names is a key feature of the flow. The current layout database, golden DRC results database, and the regression DRC results database are combined into a single layout, and the rule check results are compared using the generated XOR rules.

Calibre DESIGNrev, Calibre nmDRC-H, and Calibre RVE are the primary tools for the flow.

Generating XOR Regression Files with Calibre DESIGNrev 156

Generating XOR Regression Files with Calibre DESIGNrev

Calibre DESIGNrev can use Tcl scripts in batch mode to generate layout databases and information about layout databases.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

This procedure discusses a script that creates an OASIS layout from a current layout database, a golden DRC results database, and a regression DRC results database. The script also generates an XOR comparison rule file that compares the results in the two input results databases by basing the compared layer names on the names of rule checks. This is useful in a DRC regression flow where the layout and rule file can change, while the rule check names remain the same.

Prerequisites

- A current layout database.
- A DRC rule file where baseline rule check names have been maintained across revisions.
- A golden DRC results database that is used as the benchmark for DRC regression testing. The results are generated from baseline versions of the layout and DRC rule file.
- A regression DRC results database that is generated from the current version of the layout and DRC rule file.
- A Calibre DESIGNrev license.

Procedure

1. Open a new file called *create_xor_files* and enter this code:

```
#! /bin/sh
#################################################################
# The purpose of this script is to set up regression testing of rule
# check results. The script is called by Calibre DESIGNrev and
# outputs an SVRF rule file suitable for XOR checking of rule check
# output.
#
# Usage: <scriptname> <layout> <golden_drc_db> <regression_drc_db>
#
# layout is the pathname of the current version of the physical
# layout. golden_drc_db is the benchmark DRC results database for
# the rule file being regression tested. regression_drc_db is the
# current DRC results database produced from the current layout with
# the rules being regression tested.
#################################################################
# required to convert paths to polygons
export MGC_CWB_CONVERT_ZWPATH_TO_POLYGON=1;

# the Tcl file to be executed in Calibre DESIGNrev
drv_tmp_file=".drv.tcl";
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
# take the content of this script from the first line through the
# first line that starts with exit and delete it. then copy the
# remaining portion to drv_tmp_file.
sed -e '1,/^exit/d' $0 > $drv_tmp_file;

# in the Calibre DESIGNrev shell, execute the following:
# .drv.tcl <original_script_name> .drv.tcl <command_line_args>
calibredrv -shell $drv_tmp_file $0 $drv_tmp_file $@;

exit;

# ensure variable setting for changing zero-width paths to polygons
if ![info exists ::env(MGC_CWB_CONVERT_ZWPATH_TO_POLYGON)] {
    puts "\nERROR: env var MGC_CWB_CONVERT_ZWPATH_TO_POLYGON not \
            set\n";
    exit 1;
}
# the variable value must be 1
if {$::env(MGC_CWB_CONVERT_ZWPATH_TO_POLYGON) != 1} {
    puts "\nERROR: env var MGC_CWB_CONVERT_ZWPATH_TO_POLYGON value \
            is not 1\n";
    exit 1;
}

# from the calibredrv -shell call previously, initialize the command
# line args as variables.
foreach {script tmpscript layout gold_rdb regr_rdb} $argv break
if {[llength $argv] != 5} {
   puts "\nUsage: $script <layout> <golden rdb> <regression rdb>\n";
   exit 1;
}

##### variable setup
# xor comparison rule file name
set xor_svrf xor.svrf;
# xor DRC report file name
set xor_report "xor.report";
# output layout
set out_layout out.oas;
# layout design handle
set L [layout create $layout -dt_expand];
# physical layout primary cell name
set topcell [$L topcell];
# list of physical layer numbers in ascending order
set layer_nos [lsort -increasing -real [$L layers]];
# for each layer number, add an entry to the larr array indicating
# the layer is physical with a value of 1.
foreach layer_N $layer_nos {set larr($layer_N,is_physical) 1};
# highest physical layer number plus 1
set maxlayer [expr int([lindex $layer_nos end]) + 1];
# initialize layer names list
set layer_names [list];
# initialize rdb layer numbers list
set rdb_layer_nos [list];
# initialize XOR comparison rule file script
set svrf {};
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
##### populate the layer array with layers from the RDBs.
# the passed-in arguments are a list of layer numbers, a results
# database identifier, and a layout database file handle.
proc populate_larr {layer_nos rdbID L} {

# larr initially has physical layer numbers, but the array is then
# populated with additional information.
# the layer_names list is initially empty.
  global larr layer_names;

# process the input layer numbers and map them into the array
  foreach layer_N $layer_nos {
# if the layer number is already in the array as physical, skip it
    if [info exists larr($layer_N,is_physical)] continue;
# otherwise, place the layer number into the array as originating
# from the merged physical layout with an RDB.
    set larr($layer_N,is_physical) 0;
# get the layer name corresponding to the number.
# results databases have this information, while the physical layout
# may not.
    set name [$L layernames $layer_N];
# put the name into the layer_names variable list
    lappend layer_names $name;

# map the layer name to its originating results database and
# layer number, and place into the layer array.
    lappend larr($name,layer,$rdbID) $layer_N;
# map the layer number to the original name, and place into the
# layer array.
    set larr($layer_N,oname) $name;
# map the layer number to the originating results database and place
# into the layer array.
    set larr($layer_N,rdb) $rdbID;
# assign a layer name and results database identifier string as
# the layer name for each layer number in the merged layout handle.
    $L layernames $layer_N ${name}:$rdbID;
  }
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
##### main processing section.
# process the golden and regression results database files.
# these are used to populate the layer array.
foreach rdb "[list $gold_rdb $regr_rdb]" rdbID {1 2} {
  set Lrdb [layout create $rdb];
# merge the physical layout with the current RDB in append mode.
  set L [layout merge $L $Lrdb $maxlayer -mode append];
# reset the layer numbers to conform with the merged layout.
  set layer_nos [lsort -increasing -real [$L layers]];
# reset the maximum layer number.
  set maxlayer [expr int([lindex $layer_nos end]) + 1];
# populate the layer array larr.
  populate_larr $layer_nos $rdbID $L;
# drop the extra topcell inserted by merging.
  $L delete cell [$L topcell];
}
# sorted unique layer names across all input databases.
set sort_layer_names [lsort -unique $layer_names];

# output combined layout twice to expand paths.
$L oasisout $out_layout;
set L [layout create $out_layout -dt_expand];
$L oasisout $out_layout;
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
append svrf "
/*****************************************************************
This file includes SVRF/TVF Technology under license by
Siemens Digital Industry Software. \"SVRF/TVF Technology\" shall
mean the Standard Verification Rule Format (\"SVRF\") and
Tcl Verification Format (\"TVF\") proprietary syntaxes for
expressing process rules. You shall not use SVRF/TVF Technology
unless you are a Siemens Digital Industry Software customer as
defined by having authorized access to the password protected
support site at http://support.sw.siemens.com/. The exact terms of
your obligations and rights are governed by your respective
license. You shall not use SVRF/TVF Technology except:
(a) for your internal business purposes and (b) for use with
Calibre(r) tools. All SVRF/TVF Technology constitutes or contains
trade secrets and confidential information of Siemens Digital
Industry Software or its licensors. You shall not make SVRF/TVF
Technology available in any form to any person other than your
employees and on-site contractors, excluding Siemens competitors,
whose job performance requires access and who are under obligations
of confidentiality.
*****************************************************************/

LAYOUT PATH '$out_layout'
LAYOUT PRIMARY $topcell
LAYOUT SYSTEM OASIS
LAYOUT ERROR ON INPUT NO
DRC KEEP EMPTY NO
DRC RESULTS DATABASE 'xor.rdb' ASCII
DRC CELL NAME YES CELL SPACE XFORM
DRC MAXIMUM RESULTS ALL
DRC SUMMARY REPORT '$xor_report'
DFM DATABASE '.dfmdb' \[ALL\] OVERWRITE REVISIONS
LAYOUT USE DATABASE PRECISION YES
LAYOUT PROCESS BOX RECORD YES
LAYOUT INPUT EXCEPTION SEVERITY DATATYPE_MAP_TARGET 2
EMPTY = AREA (EXTENT) == 0

"; # end of append svrf

# get the keys corresponding to layer numbers from the
# layer array. then for those keys with a value of 0, store these as
# RDB layers.
foreach hash [array names larr "*,is_physical"] {
  set val $larr($hash);
# if the value is 1, it is a physical layer. skip those.
  if $val continue;
# get the layer numbers of the rdb layers.
  set rdb_N [lindex [split $hash ","] 0];
  lappend rdb_layer_nos $rdb_N;
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
append svrf {
// +++++++++++++++ layer settings +++++++++++++++++++++
}

# use the layer names also as check names.
foreach check $sort_layer_names {
  foreach rdbID {1 2} {
    if [info exists larr($check,layer,$rdbID)] {
       append svrf "LAYER '${check}:$rdbID' \
                    $larr($check,layer,$rdbID) \n";
    } else {
       append svrf "'${check}:$rdbID' = COPY 'EMPTY' \n";
    }
  }
}

append svrf {
// +++++++++++++++ xor rules +++++++++++++++++++++
}

foreach check $sort_layer_names {
  append svrf "'$check' \{\n";
  append svrf "\t'${check}:1' XOR '${check}:2' \n";
  append svrf "\}\n";
}

# write the XOR comparison rules.
set fid [open $xor_svrf w];
puts $fid $svrf;

puts "\n\n\nExecute XOR flow:\ncalibre -drc -hier \[turbo hyper \
            options\] $xor_svrf\n\n\n";

close $fid;
file delete $tmpscript;
exit;
```

2. Save the *create_xor_files* file.
3. Execute the following command in a shell terminal (csh assumed):

```
% ./create_xor_files <layout_path> <golden_rdb> <regression_rdb>\
|& tee log.drv
```

   where <layout_path> is the latest version of the layout, <golden_rdb> is the benchmark reference DRC results database, and <regression_rdb> is the current DRC results database for the latest version of the layout and rules. Examine the *log.drv* file to ensure the run executed properly. If there are any discrepancies, fix them and repeat this step until the run completes as expected.

Results

The output consists of two files.

The first file is *out.oas*. This is an OASIS layout database that contains layers from your current layout database and the results from both of the input DRC results databases. Layers from your

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

current layout appear in the OASIS database with the original layer names. Layers from results databases appear with layer names starting with a rule check name, followed by a colon, followed by the number 1 or 2:

<rule_check_name>:{1|2}

The suffix :1 means the layer data originates in the golden results database, and the suffix :2 means the layer data originates in the regression results database.

The second file is *xor.svrf*. This is an XOR comparison rule file that compares layers in *out.oas* that originate from the results databases. The layer naming convention is the same as for *out.oas*. The rule checks follow this general form:

```
'<rule_check_name>' {
        '<rule_check_name>:1' XOR '<rule_check_name>:2'
}
```

If the regression results database lacks rule check results that the golden results database contains, the layer definitions appear as follows:

```
EMPTY = AREA (EXTENT) == 0
...

LAYER '<rule_check_name>:1' <N>
'<rule_check_name>:2' = COPY 'EMPTY'
```

where <N> is a layer number. The correspondence appears in the reverse order if the layer is empty in the golden results database:

```
'<rule_check_name:1' = COPY 'EMPTY'
LAYER '<rule_check_name>:2' <N>
```

You can now run the XOR comparison:

```
% calibre -drc -hier -turbo xor.svrf >& log.xor
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Chapter 9
Pattern Matching Applications

Pattern matching has applications in layout analysis and layout manipulation. You can also classify results by pattern geometry.

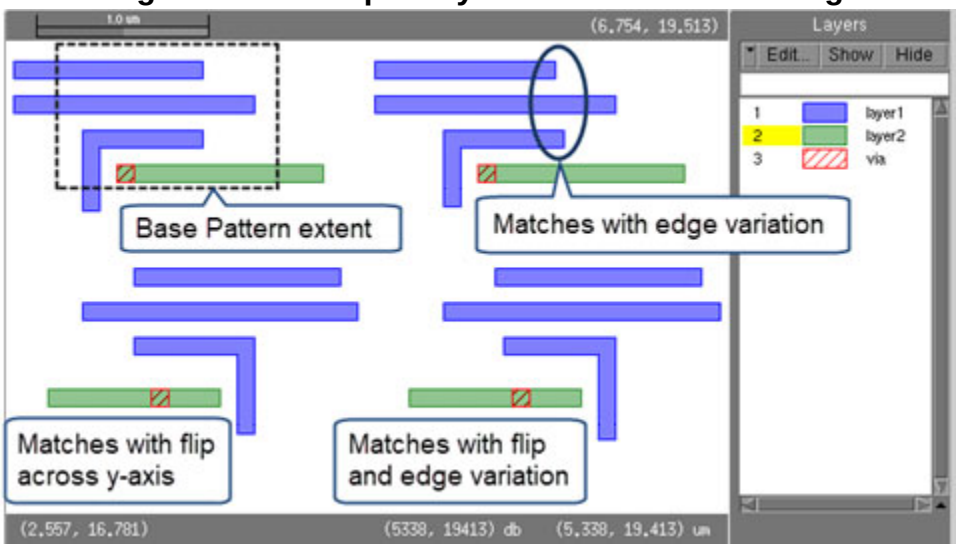
Capturing a Pattern From the Layout 165
Adding Layout Shapes Using Pattern Marker Output 168
Adding Constraints to Pattern Edges and the Output Marker 172
Converting DBCLASSIFY to DFM Pattern Classify for Result Classification 176

Capturing a Pattern From the Layout

You can easily capture a pattern from a layout using the Calibre Pattern Matching GUI. This procedure uses the drawn rectangle mode to capture a pattern that is used in the next two procedures in this chapter.

The examples in this chapter for layout editing use the following layout. Only the layer1 shapes are of interest. The location in the top left is the base pattern, which is captured in this procedure.

Figure 9-1. Example Layout for Pattern Matching



The layout has three other pattern locations:

- Lower left, the base pattern flipped across the y-axis. By default all orientations of a pattern are matched, so this pattern matches the base pattern.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Top right, the base pattern with edge variation. This pattern matches only if appropriate edge constraints are added to the base pattern.
- Lower right, the base pattern with edge variation and flip. This pattern matches only if appropriate edge constraints are added to the base pattern.

The procedure "Adding Constraints to Pattern Edges and the Output Marker" on page 172 describes how to add constraints to the pattern.

Prerequisites

- A Calibre Pattern Matching license.

Video

Capture Pattern from a Layout Open in Calibre DESIGNrev — This video demonstrates the capture process for bounding box capture mode. This mode captures pattern shapes within a rectangular region that you draw in the layout viewer.

Procedure

1. Open the layout in one of the Calibre layout viewers, such as Calibre DESIGNrev.

   Tip

   View the layout at the top cell level. The capture process clips the pattern for all levels of hierarchy at and below the currently selected cell level.

2. In the layout viewer, hide all layers except the layer of interest (layer1 in this example).
3. In the layout viewer, choose **Verification** > **Pattern Matching** > **Run Capture** to open the Capture Patterns wizard.

   Use the default library name of *lib.pmdb* for the Output File path.

4. Choose "Bounding Box" as the Capture Mode and click **Next**.
5. Define the pattern region:
   a. Click the **Select Area** button.
   b. In the layout viewer, drag the cursor to draw a rectangle around the base pattern extent, as shown in Figure 9-1.

      The cell name and rectangle coordinates populate the dialog box entries.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

6. Specify the target layers:

   a. Click **Get Visible Layers**.

      This populates the layer table with information from the open layout.

   b. Make sure the Layout Layer column has the correct layer number for the layer to capture. If not, double-click in the cell and choose the correct layout layer.

   All other options can remain at the default settings.

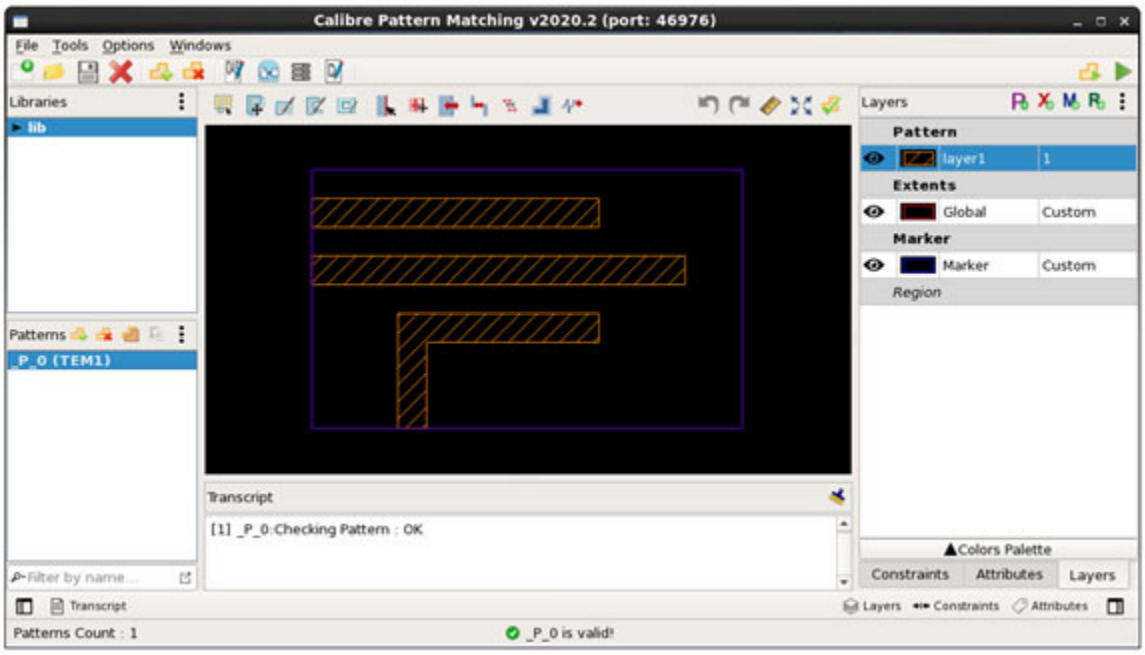
7. Click **Capture**.

   The transcript is displayed. You can view the generated rule file in the transcript.

   The Calibre Pattern Matching GUI opens the captured library.

8. Click the pattern name in the list of patterns in the lower left of the Calibre Pattern Matching GUI.

   The pattern should be similar to that in the following image.



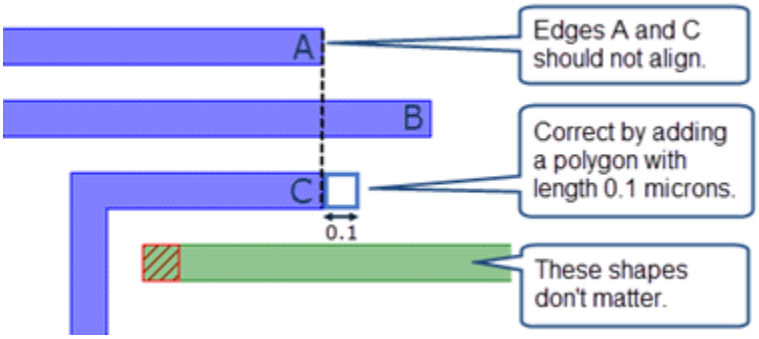
   If the pattern is not correct, click **Back** twice in the Capture Patterns wizard to redo the capture. Uncheck “Append to an existing library” to overwrite the existing library, then adjust options and recapture the pattern.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Adding Layout Shapes Using Pattern Marker Output

You can use pattern matching to identify a problem area in a layout and fix the problem by adding layout shapes.

Figure 9-2. Layout Correction with Pattern Marker Output



Pattern matching generates a polygon layer with the required layout corrections.

This procedure uses exact pattern matching for simplicity. In reality, some variation in the position of the pattern edges is usually allowed. The procedure "Adding Constraints to Pattern Edges and the Output Marker" on page 172 discusses that case.

Prerequisites

- A pattern library with a pattern that matches the area of interest. See "Capturing a Pattern From the Layout" on page 165.
- A layout in GDS or OASIS format. The layout should be suitable for testing with your pattern library.

Procedure

1. Open the pattern library in the Calibre Pattern Matching GUI:

   ```
   calibrepm lib.pmdb
   ```

2. Edit the pattern to add a custom marker layer.

   a. Select the pattern in the pattern list in the lower left of the GUI.

   b. Click the **Layers** tab at the bottom of the panel to the right of the pattern canvas.

   c. On the **Layers** tab, click the button to add a marker layer.

   In the Add Marker dialog box, change the marker name to m_edit and choose a marker type of Custom.

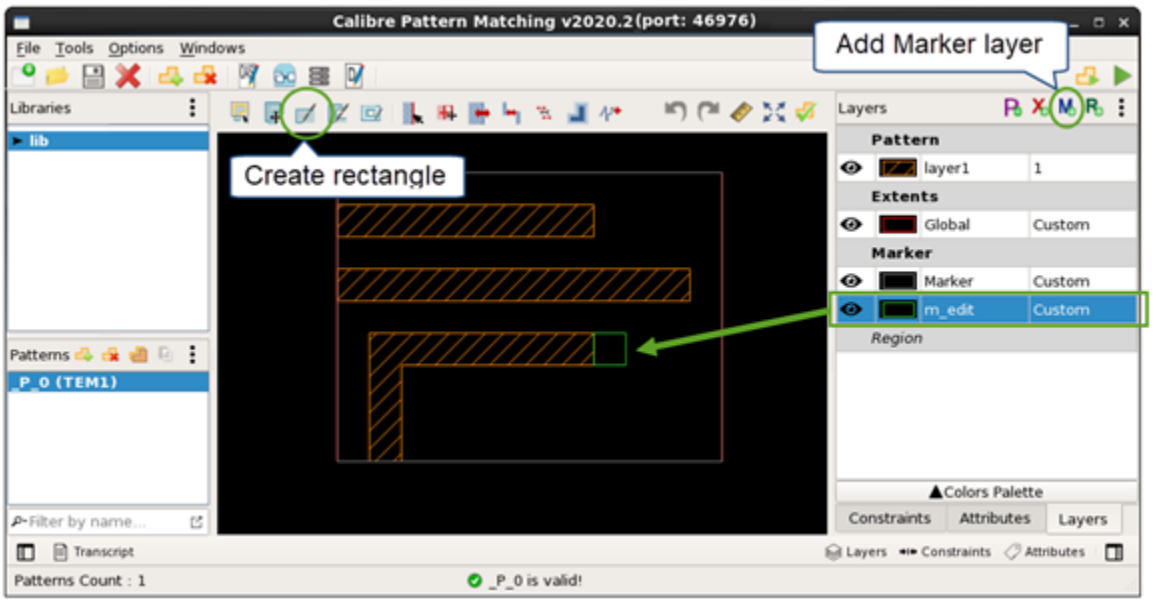
3. Add a custom marker shape to the pattern to correct the layout.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

a. In the **Layers** tab, select the m_edit marker layer.

b. Above the pattern canvas, click the button to enter Create Rectangle drawing mode.

   Use the cursor to draw a rectangle in the pattern where the added layout shape should be placed. Make the marker shape extend 0.1 microns to the right of edge C (refer to Figure 9-2).



4. Click in the top toolbar to save the library.

5. Use the GUI to run pattern matching with the library. This step creates a basic rule file that is edited in a later step.

   a. Click the button in the top right of the GUI to start a pattern matching run with the open library.

   b. Select **Layout File** as the layout source and fill in the text field with the path to the layout.

   Tip

   You can also run on an open layout by selecting **Active Layout** as the layout source. See "Running Pattern Matching From the GUI" in the *Calibre Pattern Matching User's Manual*.

   c. In the layers table, make sure the library layers correspond to the correct layer number in the layout. If not, click in the Layout Layers cell and select the correct layer number.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Other options can remain at their default settings.

d. Click **Run**.

The transcript is displayed, along with a list of the files created by the run.

6. Examine the results of the pattern matching run:

   a. Open the DRC results by clicking the folder icon in the DRC RDB row.

   The layout is opened in Calibre DESIGNrev and the DRC results database is opened in Calibre RVE.

   b. In Calibre RVE, select the pm_Marker_out check and type “H” on the keyboard to highlight all results.

   All matching layout locations should be highlighted. The highlight marker is the pattern extent.

   c. In Calibre RVE, select the pm_m_edit_out check and type “H” on the keyboard to highlight all results.

   The custom marker shapes are highlighted. Confirm that the shapes make the layout modifications that you want.

7. Edit the pattern matching rule file to save an OASIS layout file with the pm_m_edit_out custom marker shapes.

   a. In the run directory, copy the file *pm_test_lib.svrf* to the new file *edit_layout.svrf*.

   b. Open the file *edit_layout.svrf* in a text editor.

   The file includes the following lines:

```
INCLUDE <path>/lib.svrf
CMACRO pm:lib pm_layer1 " " pm_Marker_out pm_m_edit_out
```

   The CMACRO performs pattern matching with the compiled library (*lib.svrf*). The arguments to the CMACRO are the following:

   - pm:lib — References the compiled library
   - pm_layer1 — The layout layer to match
   - “ ” — CMACRO options (none in this case)
   - pm_Marker_out — Output layer from the default “Marker” output (the pattern extent)
   - pm_m_edit_out — Output layer from the custom marker “m_edit”

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

c. Add the following code to the file *edit_layout.svrf* to write out the layers of interest to an OASIS file.

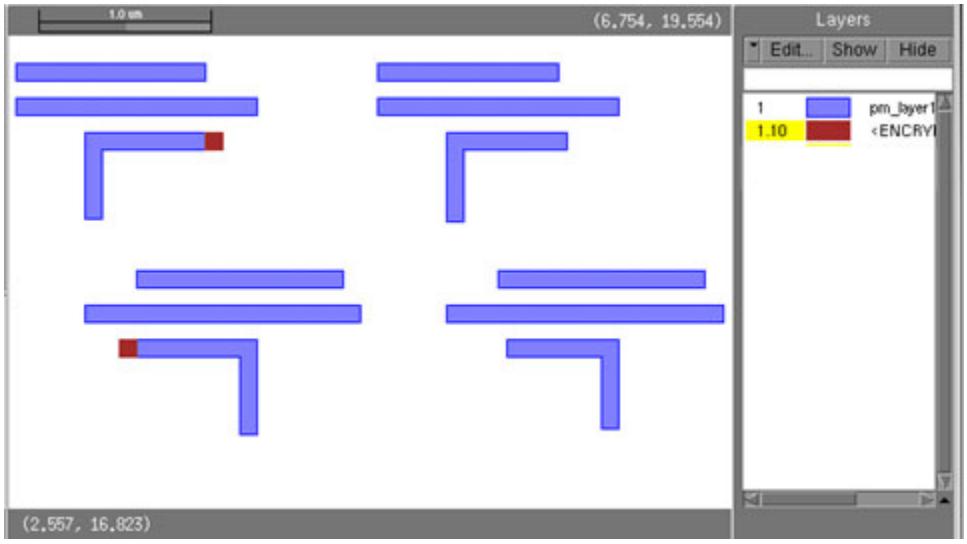
```
// optional, include other original design layers if
// you want to write them out in the OASIS file
//LAYER layer2 2
//LAYER via    3

layout_out {
DFM RDB OASIS new_layout.oas
    pm_layer1     1 0  // the layer that is matched to
    pm_m_edit_out 1 10 // custom marker output
//    layer2        2 0
//    via           3 0
}

// An OASIS file can also be written using DRC Check Map.
// pm_edit_out is both a layer and a rule check name
// DRC Check Map pm_m_edit_out OASIS 1 10 "map_layout.oas"
```

8. Run Calibre with the new rule file:

```
calibre -drc -hier edit_layout.svrf |& tee log
```

Results

For the layout shown in Figure 9-1 on page 165 with the procedure "Capturing a Pattern From the Layout", there are two matched locations, where one matched location is flipped across the y-axis. The file *new_layout.oas* looks like the following image, The two locations on the right are not exact matches to the pattern, so they are not corrected.



As part of the run, the pattern library is compiled and saved as the file *<lib>.svrf*, where *<lib>.pmdb* is the name of the pattern library. Other files are also created; see "Running Pattern Matching From the GUI" in the *Calibre Pattern Matching User's Manual*.

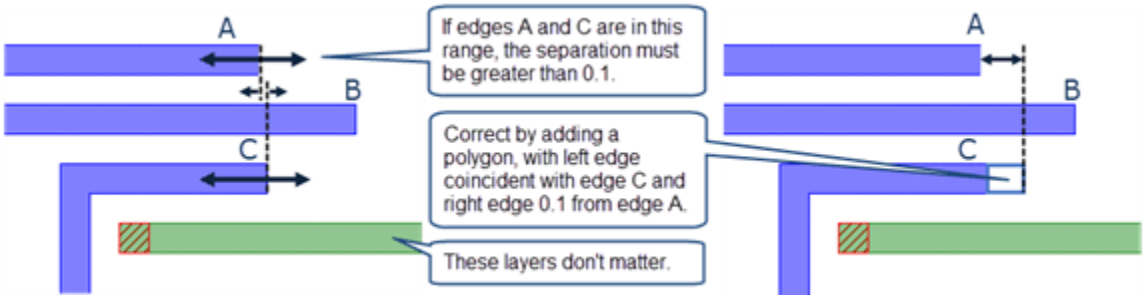
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

See "Adding Constraints to Pattern Edges and the Output Marker" on page 172 for instructions on adding constraints to this pattern so that the two layout locations on the right are matched.

Adding Constraints to Pattern Edges and the Output Marker

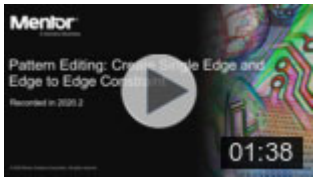
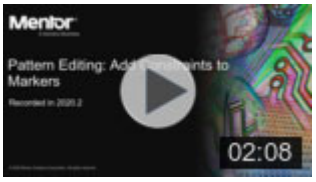
You can add constraints to a pattern to allow for variations in edge location. You can also constrain an output marker to be a fixed distance from a pattern edge that has constraints.

This procedure is similar to "Adding Layout Shapes Using Pattern Marker Output" on page 168 and assumes you have the pattern library created for that task.



Pattern matching generates a polygon layer with the required layout corrections.

These videos demonstrate the steps in this procedure:

"Pattern Editing: Create Single Edge and Edge to Edge Constraint"	"Pattern Editing: Add Constraints to Markers"
	

Prerequisites

- A pattern library with a pattern that matches the area of interest. Begin with the pattern created for "Adding Layout Shapes Using Pattern Marker Output" on page 168, which includes a custom marker.
- A layout in GDS or OASIS format. The layout should be suitable for testing with your pattern library.

Procedure

1. Open the pattern library in the Calibre Pattern Matching GUI:

```
calibrepm lib.pmdb
```

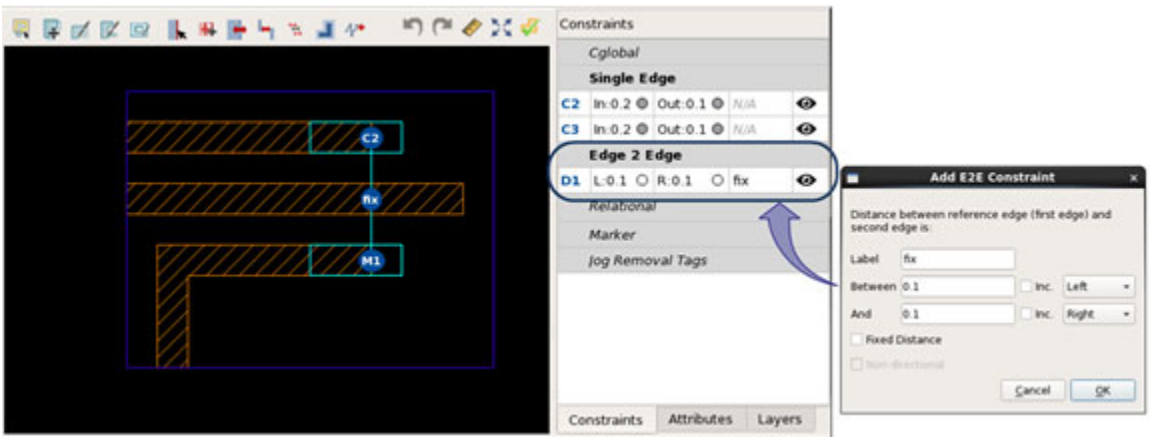

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

2. Edit the pattern to add single edge constraints to edges A and C.
   a. Select the pattern in the pattern list in the lower left of the GUI.
   b. In the toolbar above the pattern canvas, click the button to enter “Add Single Edge Constraint” mode.

      Click edge A (it will turn green), then move the cursor to the right edge of the constraint range and click, then to the left and click.

      Repeat for edge C.
   c. Click the **Constraints** tab at the bottom of the panel at the right of the GUI to view a listing of the two constraints just added.
3. Add an edge to edge constraint between edges A and C, to indicate the constraint on the edge separation.
   a. Click the button to enter “Add Edge to Edge Constraint” mode.
   b. Click edge A.
   c. Click edge C.

      The Add E2E Constraint dialog box opens. Add a label if desired.
   d. Change the constraint values to read “Between 0.1 … Left” and “And 0.1 … Right,” and uncheck the “Inc.” (inclusive) box for both values. This specifies that the second edge (or its projection) can be up to 0.1 microns to the left or right of the first edge.



4. Constrain the left edge of the custom marker to coincide with edge C:
   a. In the **Constraints** tab at the right of the GUI, click the icon to hide the C1, C2, and D1 constraints so they are not displayed. This makes it easier to see the pattern edges.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

b. Click the button to enter “Add Edge to Edge Constraint” mode.

c. Click twice at the edge C location—once on the marker edge and once on the pattern edge.

The Marker constraint M1 is entered in the Constraints tab, with parameters “Loc = e1”, indicating that the marker edge is coincident with pattern edge C.

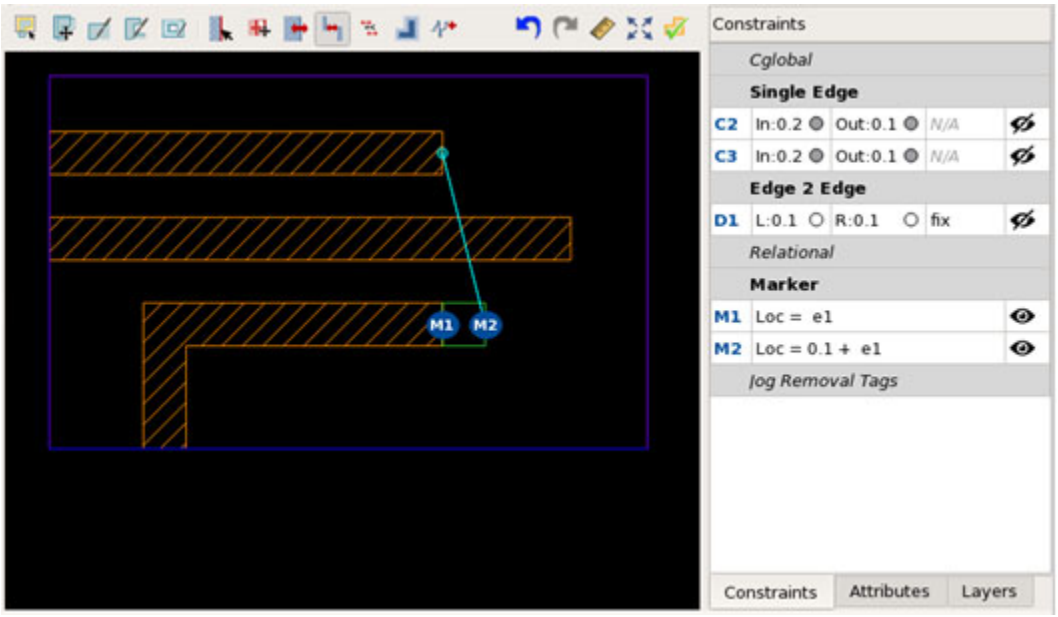
5. Constrain the right edge of the custom marker to be 0.1 microns to the right of edge A.

a. Click the button to enter “Add Edge to Edge Constraint” mode.

b. Click the right edge of the custom marker.

c. Click edge A.

The Marker constraint M2 is entered in the **Constraints** tab, with parameters “Loc = 0.1 + e1”, indicating that the marker edge must be the 0.1 microns to the right of pattern edge A.



6. Click in the top toolbar to save the library.

7. Use the GUI to run pattern matching with the library.

a. Click the button in the top right of the GUI to start a pattern matching run with the open library.

b. Select **Layout File** as the layout source and fill in the text field with the path to the layout.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Tip

You can also run on an open layout by selecting **Active Layout** as the layout source. See "Running Pattern Matching From the GUI" in the *Calibre Pattern Matching User's Manual*.

c. In the layers table, make sure the library layers correspond to the correct layer number in the layout. If not, click in the Layout Layers cell and select the correct layer number.

   Other options can remain at their default settings.

d. Click **Run**.

The transcript is displayed, along with a list of the files created by the run.

8. Examine the results of the pattern matching run:

   a. Open the DRC results by clicking the folder icon in the DRC RDB row.

   The layout is opened in Calibre DESIGNrev and the DRC results database is opened in Calibre RVE.

   b. In Calibre RVE, select the pm_m_edit_out check and type "H" on the keyboard to highlight all results.

   The custom marker shapes are highlighted. Confirm that the shapes make the layout modifications that you want.

9. (Optional) If you edited the library created with the procedure "Adding Layout Shapes Using Pattern Marker Output" on page 168, you can use the same updated rule file (*edit_layout.svrf*) to write the layers to an OASIS file.

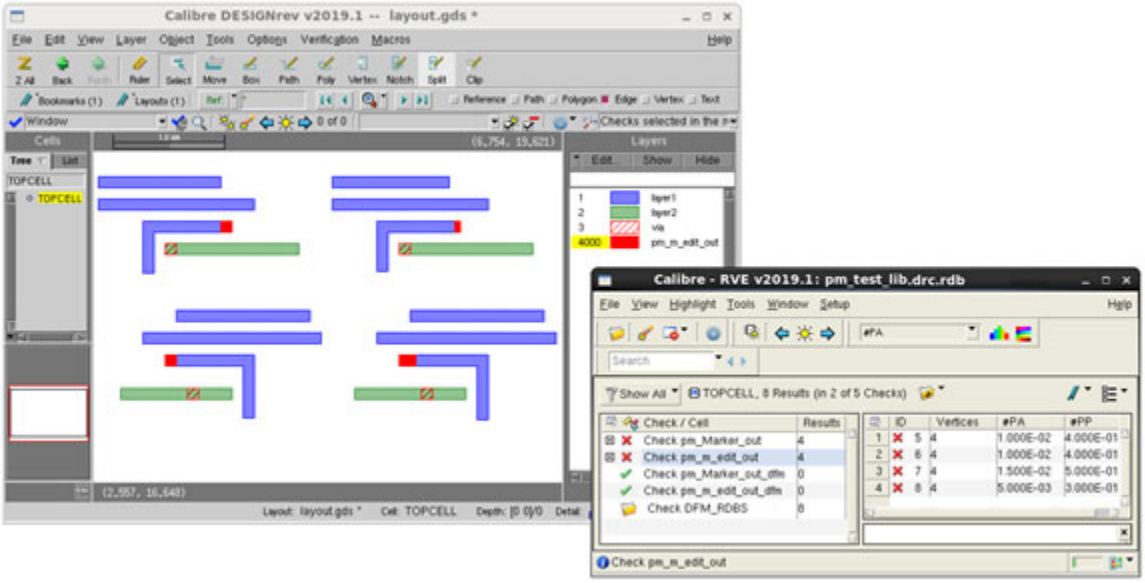
   The pattern matching run in Step 7 generates a new compiled library named *<lib>.svrf*, which is used in the pattern matching rule file.

Results

The layout shown Figure 9-1 on page 165 with the procedure "Capturing a Pattern From the Layout" has four locations that match the pattern. The two locations on the left match without constraints on the pattern edges. The two locations on the right match with constraints. The marker constraints ensure that one edge of the custom marker output is coincident with pattern

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

edge C and the other edge is 0.1 microns from pattern edge A. As a result, the length of the custom marker output varies with the position of edges A and C.



Converting DBCLASSIFY to DFM Pattern Classify for Result Classification

DBCLASSIFY is an obsolete layer operation that was used in some RET flows to classify errors (or patterns) by shape. DFM Pattern Classify can also classify error locations.

Note

As of the 2019.3 release, DBCLASSIFY no longer functions and is not documented.

The following two rule checks show a basic classify check done with each operation, where hotspot is the seed or error marker layer, and target1 and target2 are the context or pattern layers.

```
db_all {
  DBCLASSIFY FILE [
    HALO 0.4
    CENTERPT YES
  ]
  hotspot target1 target2
  MAP "ALL ERRORS"
}
```

```
pmclass_all {
  DFM PATTERN CLASSIFY
      SEED hotspot HALO 0.4
      // see note on ALLOWED_SHIFT
      ALLOWED_SHIFT 0.2
      LAYER_TARGET target1
      LAYER_TARGET target2
      CLASSIFY_PROPERTY CLASS_ID
      OUTPUT ALL
      DUP_COUNT DUP_COUNT
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note

ALLOWED_SHIFT in DFM Pattern Classify enables hotspots with slightly different center points relative to a pattern to be classified as the same pattern.

However, DFM Pattern Classify does not have an option similar to the DBCLASSIFY TOLERANCE option, so pattern geometries must be an exact match.

Use the following table to assist in converting existing DBCLASSIFY operations to DFM Pattern Classify. See DFM Pattern Classify in the *SVRF Manual* for the full syntax of the operation. A discussion of possible differences in output follows the table, then an example.

DBCLASSIFY Keyword or Setup Command	DFM Pattern Classify Equivalent
error_layer	**SEED *error_layer***
layer1 [*layerN* …]	**LAYER_TARGET *layer1*** [LAYER_TARGET *layerN*] …
MAP “UNIQUE ERRORS” (default)	OUTPUT UNIQUE
MAP “ALL ERRORS”	OUTPUT ALL (default)
MAP “XOR n”	No equivalent
Setup Commands Setup commands (TOLERANCE, for example) that do not have a DFM Pattern Classify equivalent are not listed.	
HALO *halo_value*	**HALO *halo_value*** You can also specify separate x- and y-halo values in DFM Pattern Classify with “HALO *x_halo y_halo*”.
CENTERPT YES	Default
CENTERPT NO	FROM EXTENTS
FLATTEN ONE PLACEMENT YES	OUTPUT UNIQUE NO_HIER
MATCH REFLECTIONS ROTATIONS YES (default)	MATCH_ORIENTATIONS YES (default)
MATCH REFLECTIONS ROTATIONS NO	MATCH_ORIENTATIONS NO
MATCH REFLECTIONS ROTATIONS {SYMMETRIC X-AXIS | SYMMETRIC Y-AXIS}	MATCH_ORIENTATIONS {X_AXIS | Y_AXIS}

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DBCLASSIFY Keyword or Setup Command	DFM Pattern Classify Equivalent
SUPPRESS [UNIQUE] [UNCLASSIFIED] [DUPLICATES]	The SUPPRESS setup command in DBCLASSIFY affects what is written to the RDB when RDB is specified. DFM Classify does not write RDB output, so it does not have a direct equivalent for SUPPRESS.
RDB [ONLY] filename	No direct equivalent within DFM Pattern Classify. Use the DFM RDB operation to output the derived layer to a RDB that includes property data.
default, no control	CLASSIFY_PROPERTY CLASS_ID The default class property in DFM Pattern Classify is “class”. Specify “CLASSIFY_PROPERTY CLASS_ID” to get the same class property as DBCLASSIFY.
default, no control	DUP_COUNT DUP_COUNT DBCLASSIFY includes the property DUP_COUNT by default, but only on the output for unique error locations. Specify “DUP_COUNT DUP_COUNT” to include this property on all output from DFM Pattern Classify.
no corresponding command	ALLOWED_SHIFT *value* [SEARCH_LEVEL EXACT] Recommended. This option performs auto-centering of the pattern center, with an allowed shift of *value*. The recommended value is one half of the halo size. The optional SEARCH_LEVEL EXACT keyword uses an improved search algorithm that attempts to optimize the number of duplicate patterns, and can increase runtime.

Possible Differences in Output

The number of unique, duplicate, and unclassified results from the two operations can vary for these reasons:

- DBCLASSIFY sometimes does not merge ***error_layer*** shapes that are split across the hierarchy. This causes DBCLASSIFY to classify two different error markers, while DFM Pattern Classify classifies one merged shape.
- For ***error_layer*** shapes with skew edges, small differences in the center point of the shapes can lead to different classification results. The use of ALLOWED_SHIFT *value* and SEARCH_LEVEL EXACT in DFM Pattern Classify usually prevents this.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- DFM Pattern Classify classifies all locations without a target layer edge within the halo region with a class ID of -1 (indicating no valid pattern). For example, a location with no target layer geometry and a location with a target layer geometry that completely covers and extends beyond the halo region are both classified with a class ID of -1. DBCLASSIFY classifies such locations as unique results.

Example 1

The example uses DBCLASSIFY, and writes results to an RDB. It also writes two rule check outputs, one with all results and one with unique results. DFM Pattern Classify is used to generate similar rule check output and an RDB that is similar to that produced by DBCLASSIFY.

```
LAYOUT SYSTEM OASIS
LAYOUT PATH 'layout.oas'
LAYOUT PRIMARY TOPCELL

LAYER target1  6
LAYER target2  7
LAYER hotspot  5

DRC RESULTS DATABASE drc_results
DRC SUMMARY REPORT   drc.summary

db_all {
    DBCLASSIFY FILE [
        HALO 0.4
        CENTERPT YES
        RDB dbclass.rdb
    ]
    hotspot target1 target2
    MAP "ALL ERRORS"
}

db_uniq {
    DBCLASSIFY FILE [
        HALO 0.4
        CENTERPT YES
    ]
    hotspot target1 target2
    // MAP "UNIQUE ERRORS"  // this is default
}

pmclass_all = DFM PATTERN CLASSIFY
  SEED hotspot HALO 0.4
  ALLOWED_SHIFT 0.2 // SEARCH_LEVEL EXACT
  LAYER_TARGET target1 LAYER_TARGET target2
  CLASSIFY_PROPERTY CLASS_ID
  OUTPUT ALL  // ALL is default
  DUP_COUNT DUP_COUNT
pmclass_all {
    copy pmclass_all
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
pmclass_uniq = DFM PATTERN CLASSIFY
  SEED hotspot HALO 0.4
  ALLOWED_SHIFT 0.2 // SEARCH_LEVEL EXACT
  LAYER_TARGET target1 LAYER_TARGET target2
  CLASSIFY_PROPERTY CLASS_ID
  OUTPUT UNIQUE
  DUP_COUNT DUP_COUNT
pmclass_uniq {
    copy pmclass_uniq
}

// Use DFM Property to remove invalid locations (with CLASS_ID -1)
// from pmclass_all and pmclass_uniq layers. Get unclassified locations
// with CLASS_ID = -1.
pmclass_rdb {
all_valid = DFM PROPERTY pmclass_all
            [ CLASS_ID = PROPERTY(pmclass_all, CLASS_ID)] != -1
            [ DUP_COUNT = PROPERTY(pmclass_all, DUP_COUNT)]
uniq = DFM PROPERTY pmclass_uniq
            [ CLASS_ID = PROPERTY(pmclass_uniq, CLASS_ID)] != -1
            [ DUP_COUNT = PROPERTY(pmclass_uniq, DUP_COUNT)]
// get locations of duplicates
dupl_loc = all_valid NOT uniq
dupl = DFM PROPERTY all_valid dupl_loc
            [ CLASS_ID = PROPERTY(all_valid, CLASS_ID)] != -1
            [ DUP_COUNT = PROPERTY(all_valid, DUP_COUNT)]
            [ - = COUNT(dupl_loc)] > 0
uncl = DFM PROPERTY pmclass_all
            [ CLASS_ID = PROPERTY(pmclass_all, CLASS_ID)] == -1
            [ DUP_COUNT = PROPERTY(pmclass_all, DUP_COUNT)]
DFM RDB uniq "pmclass.rdb"  ALL CELLS MAXIMUM ALL checkname "%_l_"
DFM RDB uncl "pmclass.rdb"  ALL CELLS MAXIMUM ALL checkname "%_l_"
DFM RDB dupl "pmclass.rdb"  ALL CELLS MAXIMUM ALL checkname "%_l_"
}
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Chapter 10
Calibre RVE HTML Reporting

HTML reporting in Calibre RVE for DRC creates HTML pages corresponding to views in Calibre RVE. You can specify views of DRC result tables, result highlights, histograms, and colormaps. Controls are included for grouping, filtering, and sorting the results, and for customizing the report format.

DRC HTML reporting can simplify the communication and approval process within a design team and between the design team and external groups (such as the foundry) due to the following advantages:

- HTML reports do not require manual processing once the configuration file is developed, so that the report creation process can be automated.
- HTML reports are easy to share, and do not include access to any design data. Only the layout images you want to share are included.
- HTML reports can be customized with a Cascading Style Sheet (CSS) or an HTML customization file, as described in "Output Format and Customization for DRC HTML Reports" in the *Calibre RVE User's Manual*.
- HTML reports can be automatically generated from Calibre Interactive nmDRC or Calibre RVE for DRC. When creating the report from Calibre RVE, the DRC HTML report is based on the view settings in your Calibre RVE session. See "Automatically Creating the DRC HTML Report" in the *Calibre RVE User's Manual*.

For complete information on the configuration file format and command line for DRC HTML reporting, see "DRC HTML Reporting" in the *Calibre RVE User's Manual*. The referenced section also includes an extended example and tips for troubleshooting.

DRC HTML Report for Waiver Flow **181**
DRC HTML Report for Calibre PERC LDL CD **189**
DRC HTML Report for Calibre LFD **191**

DRC HTML Report for Waiver Flow

The following example demonstrates using DRC HTML reporting to generate a report for a DRC waiver run. The report is used to document the errors that remain in the design and the waivers that were applied.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

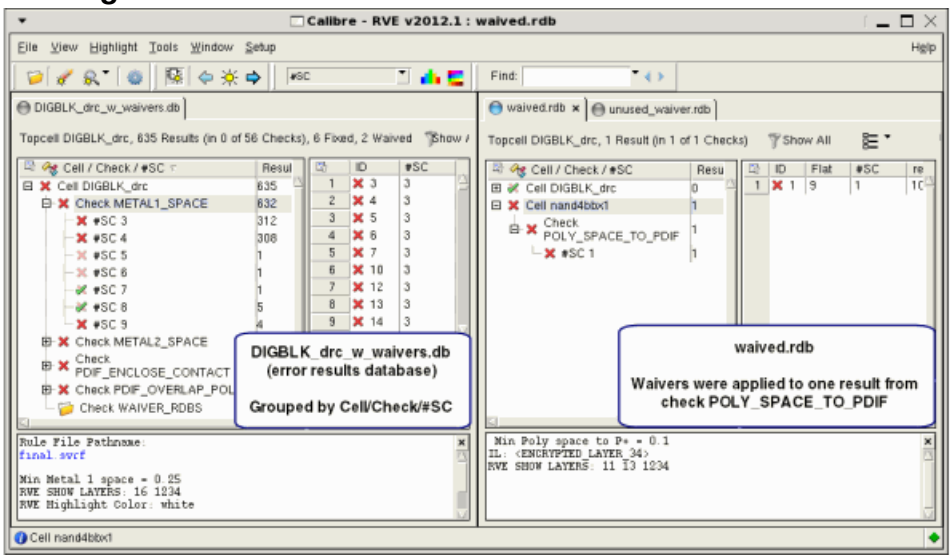
The DRC waiver run produces the following DRC results databases:

- ***DIGBLK_drc_w_waivers.db*** — The DRC error results database. Several results are marked as fixed or waived in Calibre RVE.
- ***waived.rdb*** — The waived results database.
- ***unused_waiver.rdb*** — The results database containing waivers that were not used.

See the *Calibre Auto-Waiver User's and Reference Manual* for complete information.

A view of the results in Calibre RVE is shown in the following figure, where the results databases are shown side by side.

Figure 10-1. Calibre RVE View of Waiver Flow Results



The error database (*DIGBLK_drc_w_waivers.db*) is shown on the left and the *waived.rdb* and *unused_waiver.rdb* databases are shown on the right.

The DRC HTML report described in this section is run on all databases so that the remaining errors, the applied waivers, and the unused waivers can be documented. The DRC HTML report includes the following sections:

- Summary grouped by Cell/Check
- Summary grouped by Cell/Check/#SC (shape class)
- DRC results with highlights for check METAL1_SPACE. This report is grouped by Cell/Check/#SC and sets RowCount = 1, so that only one result per shape class is seen. This report only processes the main results database by default.
- Table view of DRC results for METAL1_SPACE that are waived within Calibre RVE. This report only processes the main results database by default.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

- Highlights of the results in the Waiver RDBs. This report section includes the key "ApplyTo = Waiver" so that it only processes the two waiver RDBs (*waived.rdb* and *unused_waiver.rdb*) produced by the run.

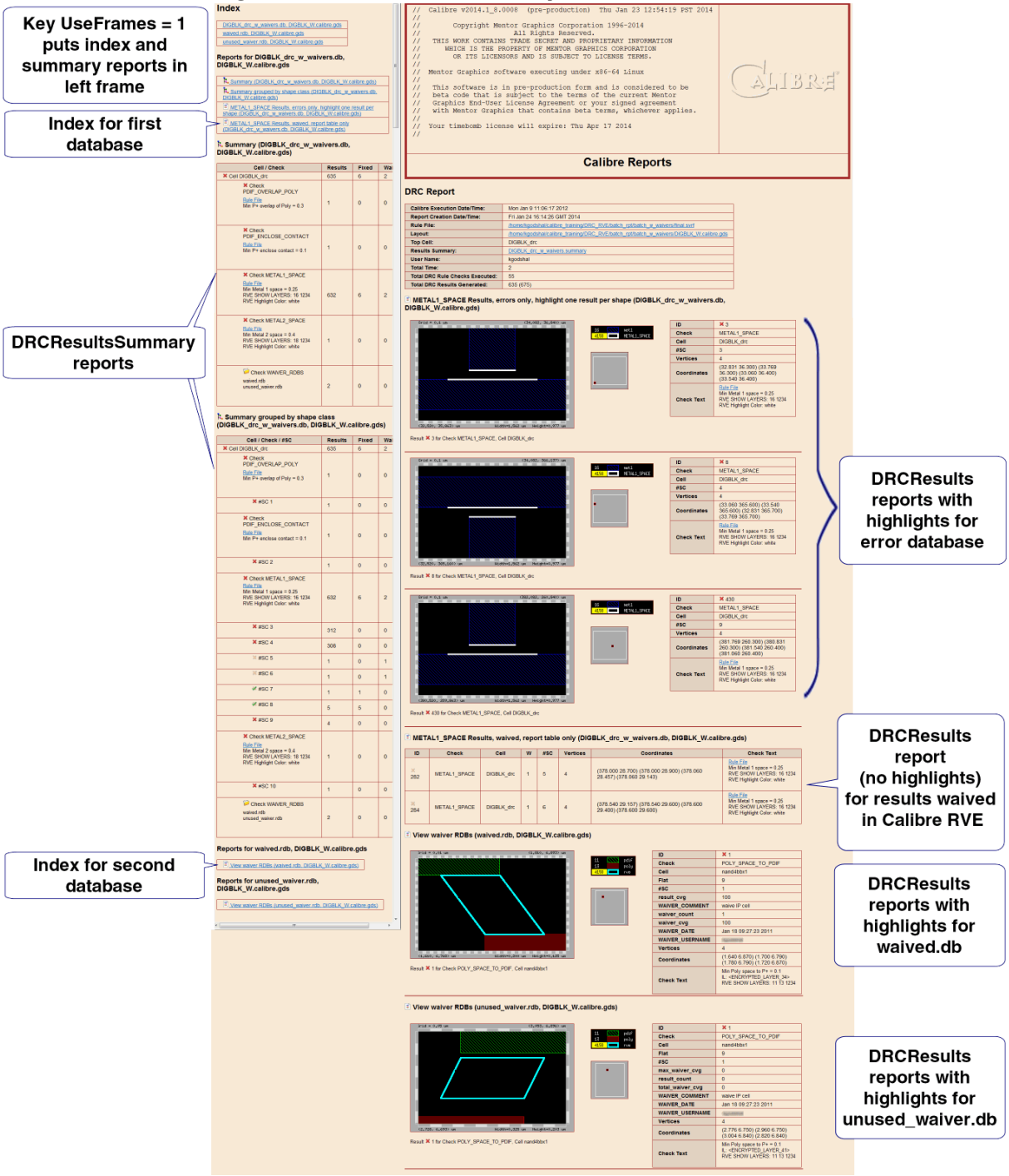
  This section produces two reports in the output; for the example shown each report includes one highlight image.

A check text override file is used to specify the layers shown during highlighting, and a DESIGNrev preferences file is used to adjust display options.

The complete HTML report is shown in Figure 10-2; details about each section are given following the figure, including the relevant sections of the configuration file.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 10-2. DRC HTML Report for Waiver Flow



Command Line

```
% calibre -rve -drc -input ./rdbSpec_all.txt
      -report ./batch_waivers_all.ini -outputdir batch_waivers_all
      -summaryfile ./DIGBLK_drc_w_waivers.summary
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Input Files

rdbSpec_all.txt — A DRC HTML Report Database Map File. This file specifies the DRC Results Database, the layout database, and a Calibre DESIGNrev layer properties file, as shown here:

```
; error database
DIGBLK_drc_w_waivers.db,,DIGBLK_W.calibre.gds,DIGBLK_W.calibre.gds.layerp
rops
```

batch_waivers_all.ini — The DRC HTML Report Configuration File. The text for this file is described in the section “Configuration File and Output Description”.

DIGBLK_drc_w_waivers.summary — A DRC Summary Report created by the Calibre run that created the results database.

Output Directory and Files

The HTML report is named *index.htm* and created in the output directory *batch_waivers_all* in this example. All supporting files are located in the output directory.

Configuration File and Output Description

Each section in the configuration file is described in the following section and the resulting report section is shown. The complete configuration file is included at the end of this example.

Common section

This example common section specifies the directory for a Calibre DESIGNrev *preferences* file, the layout image size, and other general settings.

```
[Common]
UseFrames = 1
DrvOptionsFilePath = ../
LayoutWidth = 200
LayoutHeight = 250
WorldView = 1
```

For the views shown here, the *preferences* file contains one line:

```
prefs_grid_style -1
```

This setting turns off grid markers in the layout images. This *preferences* file is merged with the defaults used by DRC HTML reporting, with settings in the specified file taking precedence if there is a conflict.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

DRCResultsSummary report section

This example includes two sections that produce a DRCResultsSummary report. The first summary is grouped by Cell/Check, the second by Cell/Check/#SC (shape class):

```
[Summary]
ReportType = DRCResultsSummary
GroupBy = Cell/Check

[Summary grouped by shape class]
ReportType = DRCResultsSummary
GroupBy = Cell/Check/#SC
```

The view for the second report is shown in the following image and corresponds to the tree view shown on the left side of Figure 10-1.

Summary grouped by shape class (DIGBLK_drc_w_waivers.db, DIGBLK_W.calibre.gds)

Cell / Check / #SC	Results	Fixed	Waived
Cell DIGBLK_drc	635	6	2
Check PDIF_OVERLAP_POLY Rule File Min P+ overlap of Poly = 0.3	1	0	0
#SC 1	1	0	0
Check PDIF_ENCLOSE_CONTACT Rule File Min P+ enclose contact = 0.1	1	0	0
#SC 2	1	0	0
Check METAL1_SPACE Rule File Min Metal 1 space = 0.25 RVE SHOW LAYERS: 16 1234 RVE Highlight Color: white	632	6	2
#SC 3	312	0	0
#SC 4	308	0	0
#SC 5	1	0	1
#SC 6	1	0	1
#SC 7	1	1	0
#SC 8	5	5	0
#SC 9	4	0	0
Check METAL2_SPACE Rule File Min Metal 2 space = 0.4 RVE SHOW LAYERS: 18 1234 RVE Highlight Color: white	1	0	0
#SC 10	1	0	0
Check WAIVER_RDBS waived.rdb unused_waiver.rdb	2	0	0

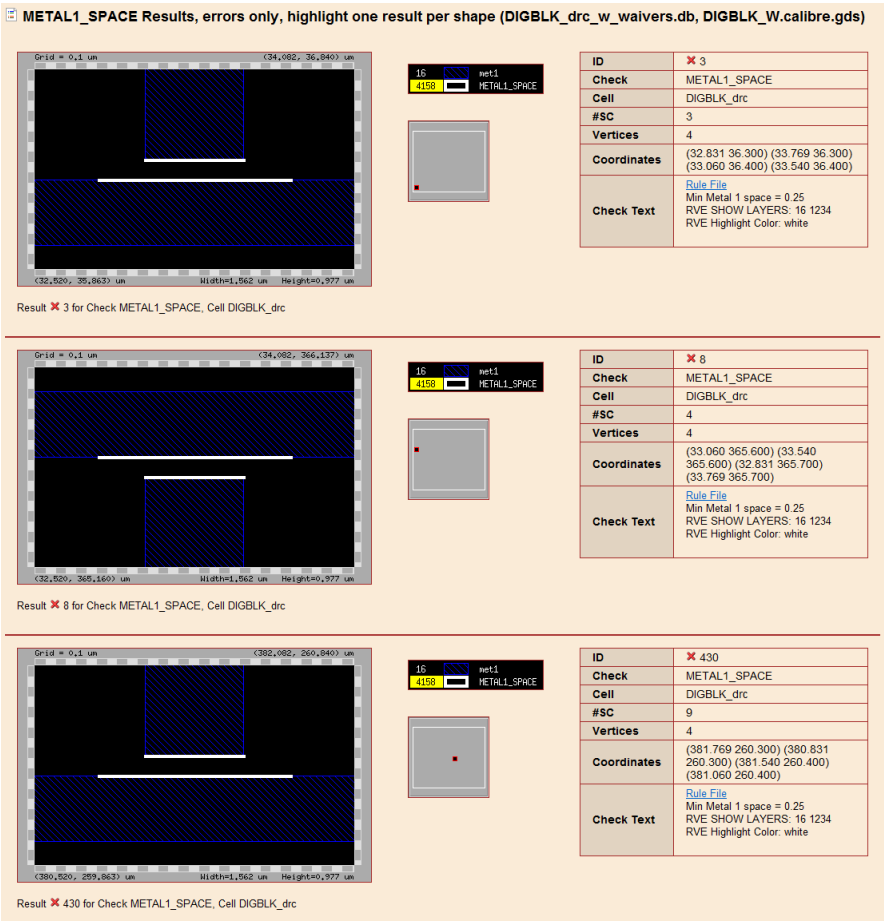
DRCResults report section

The first DRCResults report section is grouped by Cell/Check/#SC and only includes results for the METAL1_SPACE check. The intent of this report is to show one highlight for each of the three result shapes that are not waived or fixed.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
; group by shape class,
; set RowCount=1 to get just one highlight for each result shape
[METAL1_SPACE Results, errors only, highlight one result per shape]

ReportType = DRCResults
GroupBy = Cell/Check/#SC
OrCheckFilter = METAL1_SPACE
; view errors only
WaivedStateFilter = NotWaived
FixedStateFilter = NotFixed
RowCount = 1
Highlight = 1
ZoomLayoutFactor = 0.6
```

The report section is shown in the following figure:



The second DRCResults report section does not include highlights and selects only waived results. By default, this section is only applied to the main results database, which is of RDB type “General”.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
[METAL1_SPACE Results, waived, report table only]
ReportType = DRCResults
GroupBy = Cell/Check/#SC
OrCheckFilter = METAL1_SPACE
WaivedStateFilter = Waived
```

The report section is shown in the following figure:

METAL1_SPACE Results, waived, report table only (DIGBLK_drc_w_waivers.db, DIGBLK_W.calibre.gds)

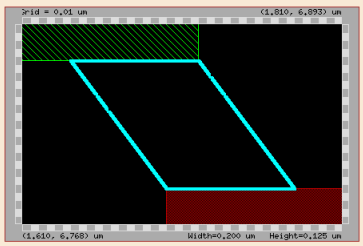
ID	Check	Cell	W	#SC	Vertices	Coordinates	Check Text
✖ 282	METAL1_SPACE	DIGBLK_drc	1	5	4	(378.000 28.700) (378.000 28.900) (378.060 28.457) (378.060 29.143)	Rule File Min Metal 1 space = 0.25 RVE SHOW LAYERS: 16 1234 RVE Highlight Color: white
✖ 284	METAL1_SPACE	DIGBLK_drc	1	6	4	(378.540 29.157) (378.540 29.600) (378.600 29.400) (378.600 29.600)	Rule File Min Metal 1 space = 0.25 RVE SHOW LAYERS: 16 1234 RVE Highlight Color: white

The last DRCResults report section highlights the waivers found in the *waived.rdb* and *unused_waivers.rdb* databases produced during a DRC waiver run. The ApplyTo key is used to apply this section to only Waiver RDBs.

```
; ; highlight applied and unused waivers using ApplyTo = Waiver
[View waiver RDBs]
ReportType = DRCResults
ApplyTo = Waiver
GroupBy = Cell/Check
Highlight = 1
```

This section produces the output shown in the following figure:

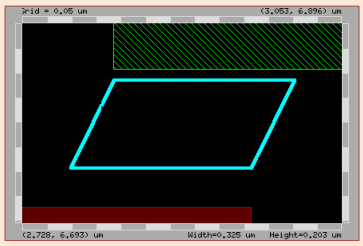
View waiver RDBs (waived.rdb, DIGBLK_W.calibre.gds)



Result ✖ 1 for Check POLY_SPACE_TO_PDIF, Cell nand4bbx1

ID	✖ 1
Check	POLY_SPACE_TO_PDIF
Cell	nand4bbx1
Flat	9
#SC	1
result_cvg	100
WAIVER_COMMENT	waive IP cell
waiver_count	1
waiver_cvg	100
WAIVER_DATE	Jan 18 09:27:23 2011
WAIVER_USERNAME	[illegible]
Vertices	4
Coordinates	(1.640 6.870) (1.700 6.790) (1.780 6.790) (1.720 6.870)
Check Text	Min Poly space to P+ = 0.1 IL: <ENCRYPTED_LAYER_34> RVE SHOW LAYERS: 11 13 1234

View waiver RDBs (unused_waiver.rdb, DIGBLK_W.calibre.gds)



Result ✖ 1 for Check POLY_SPACE_TO_PDIF, Cell nand4bbx1

ID	✖ 1
Check	POLY_SPACE_TO_PDIF
Cell	nand4bbx1
Flat	9
#SC	1
max_waiver_cvg	0
result_count	0
total_waiver_cvg	0
WAIVER_COMMENT	waive IP cell
WAIVER_DATE	Jan 18 09:27:23 2011
WAIVER_USERNAME	[illegible]
Vertices	4
Coordinates	(2.776 6.750) (2.960 6.750) (3.004 6.840) (2.820 6.840)
Check Text	Min Poly space to P+ = 0.1 IL: <ENCRYPTED_LAYER_41> RVE SHOW LAYERS: 11 13 1234

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Complete Configuration File

```
[Common]
UseFrames = 1
DrvOptionsFilePath = ../
LayoutWidth = 200
LayoutHeight = 250
WorldView = 1

[Summary]
ReportType = DRCResultsSummary
GroupBy = Cell/Check

[Summary grouped by shape class]
ReportType = DRCResultsSummary
GroupBy = Cell/Check/#SC

; group by shape class, then set RowCount=1 to get just one highlight for
; each result shape
[METAL1_SPACE Results, errors only, highlight one result per shape]
ReportType = DRCResults
GroupBy = Cell/Check/#SC
OrCheckFilter = METAL1_SPACE
WaivedStateFilter = NotWaived
FixedStateFilter = NotFixed
RowCount = 1
Highlight = 1
ZoomLayoutFactor = 0.6

[METAL1_SPACE Results, waived, report table only]
; ; applies to General RDBs by default
ReportType = DRCResults
GroupBy = Cell/Check/#SC
OrCheckFilter = METAL1_SPACE
WaivedStateFilter = Waived

; ; highlight applied and unused waivers using ApplyTo = Waiver
[View waiver RDBs]
ReportType = DRCResults
ApplyTo = Waiver
GroupBy = Cell/Check
Highlight = 1
```

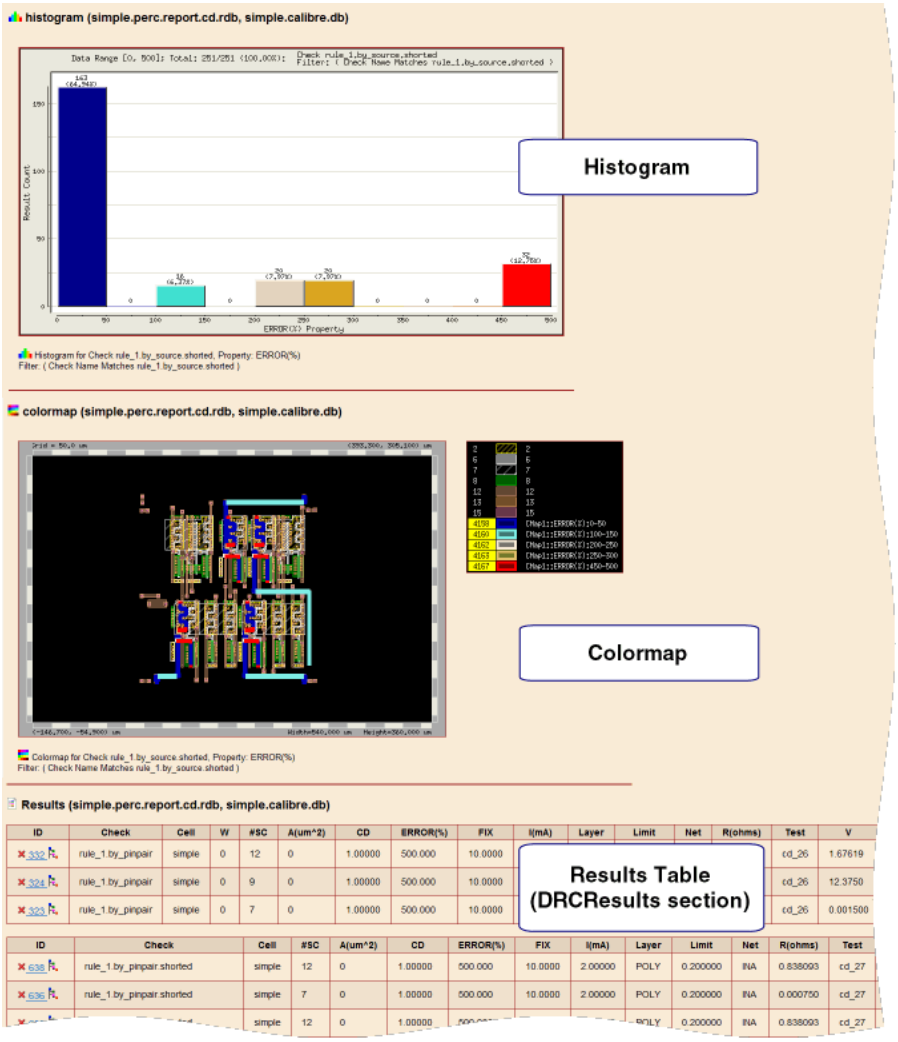
DRC HTML Report for Calibre PERC LDL CD

The following example shows a DRC HTML report created for a Calibre PERC LDL CD results database.

The report creates a histogram and colormap of the ERROR(%) property for one check. The report also includes a table of results grouped by check name, then sorted by the ERROR(%) value, with only the results with the three highest values of ERROR(%) shown for each check. A portion of the report is shown in Figure 10-3.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 10-3. DRC HTML Report for PERC LDL CD



The following sections describe the command line and input files used to create the report. In contrast to Calibre RVE for PERC, you must process the RDB created by the PERC LDL run; you cannot process the DFM database produced by the run.

Command Line

```
calibre -rve -drc -input simple_batch_path.txt -report rpt_perc_cd.ini
      -outputdir batch_out
```

Input Files

simple_batch_path.txt — A DRC HTML Report Database Map File. This file specifies the DRC Results Database, the layout database, and a Calibre DESIGNrev layer properties file, as shown here:

```
simple.perc.report.cd.rdb,,simple.calibre.db,simple.calibre.db.layerprops
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

rpt_perc_cd.ini — The DRC HTML Report Configuration File. The text for this file is described in the section “Configuration File”.

simple.perc.report.cd.rdb — A DRC Summary Report created by the Calibre run that created the results database.

Output Directory and Files

The HTML report is named *index.htm* and created in the output directory *batch_out* in this example. All supporting files are located in the output directory.

Configuration File

```
[Common]
;DrvOptionsFilePath = ../
;UseFrames = 1
;
[Summary]
ReportType = DRCResultsSummary
GroupBy = Check

[histogram ]
ReportType = DRCHistogram
GroupBy = Check
Property = ERROR(%)
OrCheckFilter = rule_1.by_source.shorted

[colormap ]
ReportType = DRCColormap
GroupBy = Check
Property = ERROR(%)
OrCheckFilter = rule_1.by_source.shorted

[Results]
ReportType = DRCResults
GroupBy = [Summary]
SortBy = ERROR(%)
; include only three highest ERROR(%) values for each check in the table
SortOrder = Descending
RowCount = 3
```

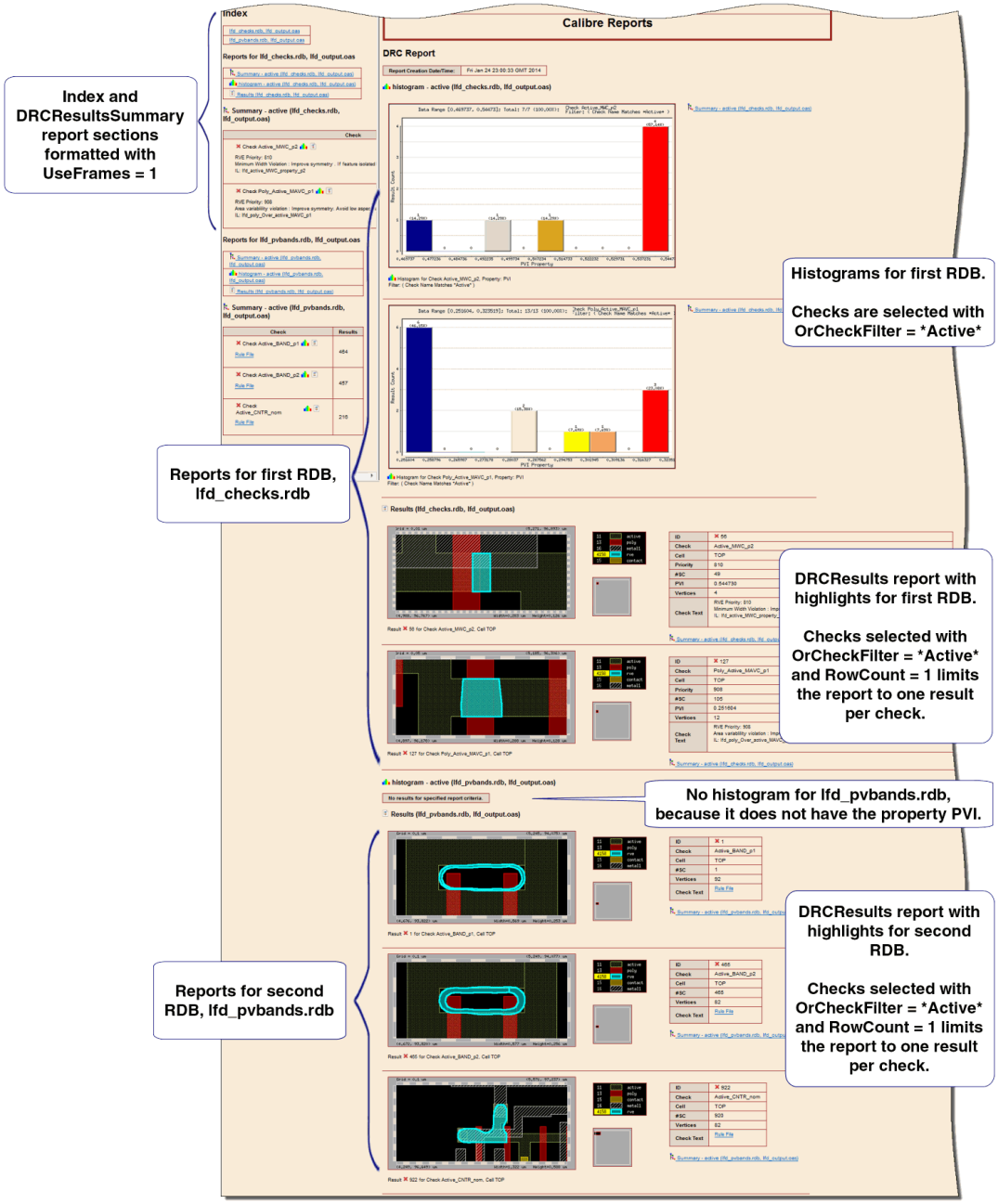
DRC HTML Report for Calibre LFD

The following example shows a DRC HTML report created for Calibre LFD (Litho Friendly Design) results.

The report processes two results databases (RDBs), and selects only results involving active layers. The report creates a histogram of the PVI (Process Variability Index) property for the selected checks, and highlights one result per selected check in each results database. The number of highlights is limited for brevity. The report is shown in Figure 10-4.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 10-4. DRC HTML Report for Calibre LFD



The following sections describe the command line and input files used to create the report.

Command Line

```
calibre -rve -drc -input lfd_output_map.txt -report rpt_lfd.ini
      -outputdir batch_out
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Input Files

lfd_output_map.txt — A DRC HTML Report Database Map File. This file specifies the results databases, the layout database, and a Calibre DESIGNrev layer properties file, as shown here:

```
lfd_checks.rdb,,lfd_output.oas,lfd_output.oas.layerprops
lfd_pvbands.rdb,,lfd_output.oas,lfd_output.oas.layerprops
```

rpt_lfd.ini — The DRC HTML Report Configuration File. The text for this file is described in the section “Configuration File”.

Output Directory and Files

The HTML report is named *index.htm* and created in the output directory *batch_out* in this example. All supporting files are located in the output directory.

Configuration File

```
[Common]
DrvOptionsFilePath = ../
UseFrames = 1
LayoutHeight = 200
LayoutWidth = 250
ZoomLayoutFactor = .5
WorldView = 1

[Summary - active]
ReportType = DRCResultsSummary
GroupBy = Check
; only include *Active* checks
OrCheckFilter = *Active*

[histogram - active]
ReportType = DRCHistogram
GroupBy = [Summary - active]
Property = PVI

[Results]
ReportType = DRCResults
GroupBy = [Summary - active]
Highlight = 1
RowCount = 1
; turn off display of Coordinates column
ShowColumns = Coordinates 0
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Chapter 11
Device Counts

There are a number of methods for determining device counts that employ various Calibre tools. These counts can be determined from the source design using Calibre PERC, and from the layout design in your DRC or LVS flow. Calibre Query Server can provide the counts after LVS has been run.

Flat Device Counts From Rule Checks 195
Device Counts in Circuit Extraction and Calibre PERC 196

Flat Device Counts From Rule Checks

You can obtain flat device counts from DRC-style rule checks. Flat counts include physical, non-reduced devices.

There are numerous layer operations that can be used for this purpose. This section shows commonly-used ones. To get device counts from rule checks, you need to know the seed layers from devices of interest.

Seed layers are most easily obtained from Device statements in the circuit extraction rule file, like this:

```
DEVICE mp (PMOS_VTL) pgate gate (G) psrcdrn (S) psrcdrn (D) nwell …
DEVICE mn (NMOS_VTL) ngate gate (G) nsrcdrn (S) nsrcdrn (D) psub …
DEVICE mp (PMOS_VTH) hvtpgate gate (G) psrcdrn (S) psrcdrn (D) nwell …
DEVICE mn (NMOS_VTH) hvtngate gate (G) nsrcdrn (S) nsrcdrn (D) psub …
```

The first layer shown in these statements is the seed layer that corresponds to the type and model (or subtype) of the device. If you do not have a circuit extraction rule file, then you will need to look at the layer derivation rules to determine what each seed layer is. With that information, you can proceed to write rule checks to determine the device counts.

As shown previously, MP devices of the PMOS_VTL model correspond to the pgate layer. The Copy operation can find devices per layer. DRC Check Map places the devices into their own results database.

```
// find the MP(PMOS_VTL) devices
MP_PMOS_VTL {COPY pgate}
DRC CHECK MAP MP_PMOS_VTL ASCII devices.db MAXIMUM RESULTS ALL

DRC SUMMARY REPORT drc.report HIER
```

The DFM Copy operation is also useful for this application and has some additional features.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

For the preceding check, the *devices.db* results are hierarchical if the run is hierarchical. For a hierarchical run, you get the flat device counts from the DRC Summary Report as shown here:

```
--- RULECHECK RESULTS STATISTICS
---
...
RULECHECK MP_PMOS_VTL ............ TOTAL Result Count = 410       (125260)
```

The number in parentheses is the flat device count for the check.

The DFM Property and DFM RDB operations offer the ability to get the flat counts into a results database as properties; hence, you would not need to look at a report. This example groups MN and MP devices by type:

```
chip = EXTENT DRAWN

mn = DFM PROPERTY chip ngate hvtngate OVERLAP ABUT ALSO
      [mn_count=count(ngate)+count(hvtngate)] > 0
mp = DFM PROPERTY chip pgate hvtpgate OVERLAP ABUT ALSO
      [mp_count=count(pgate)+count(hvtpgate)] > 0

dfm_devices {DFM RDB mn devices.rdb mp NOPSEUDO ALL CELLS ABUT ALSO
 CHECKNAME "devices"}
```

The *devices.rdb* file will contain the mn_count and mp_count properties, which are the flat device counts for MN and MP devices, respectively.

If your run is comprised only of rule checks that find devices, you can get the total device count in the run transcript from this line:

```
--- TOTAL RESULTS GENERATED = 410 (125260)
```

This shows the hierarchical and flat counts. If the run is flat, only the flat count appears.

Related Topics

Device Counts in Circuit Extraction and Calibre PERC

Device Counts in Circuit Extraction and Calibre PERC

Flat device counts can be obtained from a circuit extraction run through ERC checks, the circuit extraction report, the Query Server, or Calibre PERC rule checks.

ERC checks can be run as a part of circuit extraction (as in calibre -spice). The checks discussed under “Flat Device Counts From Rule Checks” on page 195 can also be run as ERC checks.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Additionally, the Device Layer operation can be used in an ERC check to find devices by type and subtype, like this:

```
// find the MP(PMOS_VTL) devices
MP_PMOS_VTL {DEVICE LAYER MP(PMOS_VTL)}

// or find all MP devices
// "MP" {DEVICE LAYER MP}

ERC SELECT CHECK MP_PMOS_VTL
ERC RESULTS DATABASE erc.db
ERC SUMMARY REPORT erc.report HIER
```

For the preceding check, the *erc.db* results are hierarchical if the run is hierarchical. For a hierarchical run, you can get the flat device count from an ERC Summary Report as shown here:

```
RULECHECK MP_PMOS_VTL ........... TOTAL Result Count = 410       (125260)
```

The number in parentheses is the flat device count for the check.

A circuit extraction run (which can also perform ERC) produces a layout SPICE netlist. In the top-level subcircuit definition, there is an FDC annotation in a comment, like this:

```
.SUBCKT top gnd vdd vddio vdda out_o clka_i vref_i in_i clk_i reset_i
** N=1164 EP=10 IP=2703 FDC=248921
```

This is the total flat device count of the design.

If you specify a Mask SVDB Directory QUERY statement in your circuit extraction rule file, you can use the Query Server qs::device_templates_used or DEVICE TEMPLATES USED

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

commands to find the hierarchical and flat device counts by type and subtype. Here is an example session from the standard Query Server shell:

```
% calibre -query svdb
...
INITIATING HDB QUERY SERVER:
--------------
NOTE: Cross reference commands will be unavailable since a valid cross
reference (XDB) database was not found.
   OK: Ready to serve.
DEVICE TEMPLATES USED
Device_Templates_Used 2000
Templates:
0 0 8 Oct 15 12:11:13 2013
mp(pmos_vtl) 0 410 (125260) <--final two numbers are hier. and flat counts
mn(nmos_vtl) 1 334 (121946)
mp(pmos_thkox) 4 91 (729)
mn(nmos_thkox) 5 147 (719)
c 6 2 (34)
r(rp) 7 3 (37)
r 8 2 (180)
END OF RESPONSE
0 0 0 Oct 15 12:11:13 2013
```

(You can ignore the note about the lack of an XDB database because running LVS comparison is not necessary to get these results.)

For another connectivity-based solution, you can get device counts by using Calibre PERC. See the DEVICE_COUNT high-level check or Example 2 under perc::check_data in the *Calibre PERC User's Manual* for rule check code.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Chapter 12
Electrical Rule Checks

Many Electrical Rule Check (ERC) problems can be solved using Calibre PERC. Detection of ERC problems includes Electrostatic Discharge (ESD) checks, Electrical Overstress (EOS) checks, multiple power domain crossing checks, voltage dependency checks, point-to-point current density and resistance checks, among other circuit reliability verification checks. It is advantageous to check for ERC issues in your source netlist to prevent problems in the layout and in the manufactured design. Most of these checks can also be performed on a layout netlist. Checks requiring parasitic resistance information can only be performed on the layout.

For complete information on the Calibre tool used for netlist-based circuit reliability checking, see the *Calibre PERC User's Manual*.

For working code examples, see the *shared/examples/calibre_perc* directory in your Calibre software tree.

MOSFET Bulk Pin Connections 200
MOSFET Power Domain Connections 201
Back-to-Back Diode Ground Domain Separation 202
ESD Device Protection of I/O Pads 203
ESD Resistor Protection of Gates 204
Pin Voltage Checks 205
Current Density and Resistance Checks 206

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

MOSFET Bulk Pin Connections

A common best practice is for the bulk terminals of MOS devices to be connected to voltage sources. PMOS bulk pins should be connected to power and NMOS bulk pins to ground.

Figure 12-1 shows good and bad schematic configurations.

Figure 12-1. MOSFET Bulk Pin Connections

The details for writing a rule check to detect the bad configurations in your netlist are presented in the *shared/examples/calibre_perc/PERC_Catalog/AERC/test_case3* directory in your Calibre software tree.

Related Topics

Electrical Rule Checks

ESD Resistor Protection of Gates

Calibre PERC Topology Rule Checks [Calibre PERC User's Manual]

Set Up and Run Calibre PERC [Calibre PERC User's Manual]

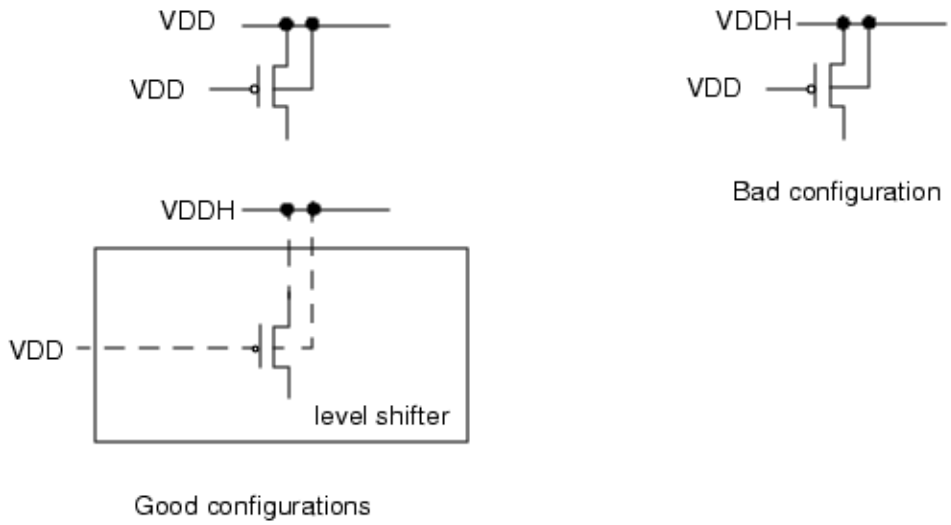
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

MOSFET Power Domain Connections

In cases where multiple power domains are present, a given MOSFET's terminals often should not be connected to different power domains, unless the device is located inside a level shifter cell.

Figure 12-2 shows good and bad schematic configurations.

Figure 12-2. MOSFET Connections to Differing Power Domains



The details for writing a rule check to detect the bad configuration in your netlist are presented in the *shared/examples/calibre_perc/PERC_Catalog/MPD/test_case1* directory in your Calibre software tree.

Related Topics

Electrical Rule Checks

Calibre PERC Topology Rule Checks [Calibre PERC User's Manual]

Set Up and Run Calibre PERC [Calibre PERC User's Manual]

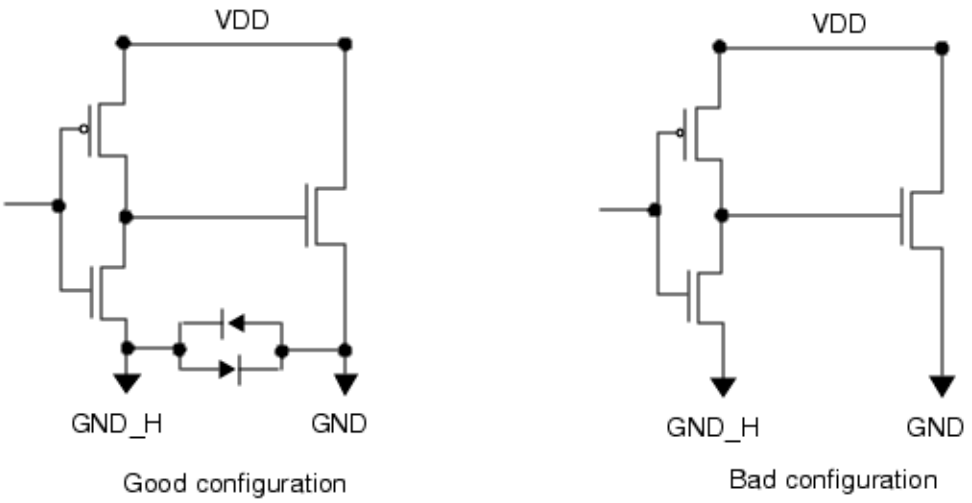
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Back-to-Back Diode Ground Domain Separation

In cases where multiple ground domains are present, signal lines that cross ground domains require back-to-back diode protection.

Figure 12-3 shows good and bad schematic configurations.

Figure 12-3. Back-to-Back Diode Separation



The details for writing a rule check to detect the bad configuration in your netlist are presented in the *shared/examples/calibre_perc/PERC_Catalog/MPD/test_case6* directory in your Calibre software tree.

Related Topics

Electrical Rule Checks

Calibre PERC Topology Rule Checks [Calibre PERC User's Manual]

Set Up and Run Calibre PERC [Calibre PERC User's Manual]

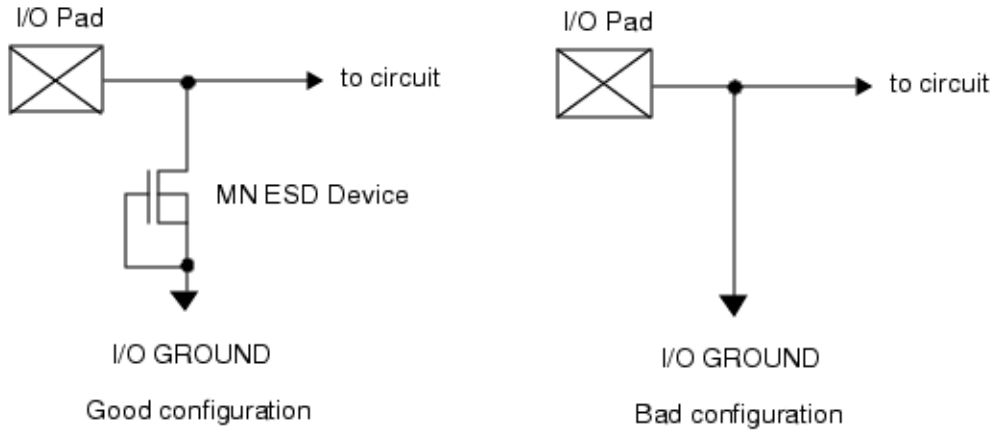
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ESD Device Protection of I/O Pads

ESD devices are frequently required to prevent damage due to voltage accumulation across I/O pads.

Figure 12-4 shows the good and bad schematic configurations.

Figure 12-4. I/O Pad With ESD Device Protection



The details for writing a rule check to detect the bad configuration in your netlist are presented under "Example: ESD Device Protection of I/O Pads" in the *Calibre PERC User's Manual*.

Also see the *shared/examples/calibre_perc/PERC_Catalog/ESD/test_case5* directory in your Calibre software tree for a related example.

Related Topics

Electrical Rule Checks

ESD Resistor Protection of Gates

Calibre PERC Topology Rule Checks [Calibre PERC User's Manual]

Set Up and Run Calibre PERC [Calibre PERC User's Manual]

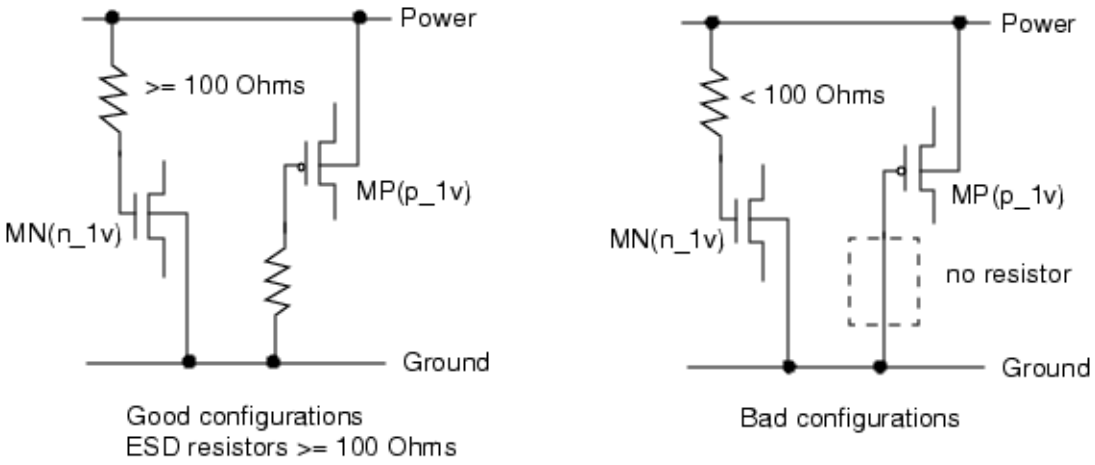
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

ESD Resistor Protection of Gates

Resistors are frequently used to protect gates tied directly to power or ground.

In this case, low-voltage gates tied directly to power or ground must have a 100 Ohm or greater resistor connected to their input:

Figure 12-5. ESD Resistor Protection of Gates



The details for writing a rule check to detect the bad configurations in your netlist are presented under "Example: ESD Resistor Protection of Gates" in the *Calibre PERC User's Manual*.

Also see the *shared/examples/calibre_perc/PERC_Catalog/ESD/test_case3* directory in your Calibre software tree.

Related Topics

Electrical Rule Checks

ESD Device Protection of I/O Pads

Calibre PERC Topology Rule Checks [Calibre PERC User's Manual]

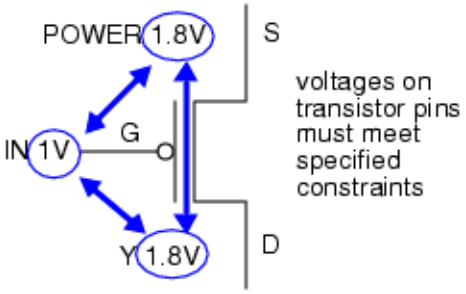
Set Up and Run Calibre PERC [Calibre PERC User's Manual]

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Pin Voltage Checks

Frequently it is desirable to check device pins to determine whether proper voltages exist for certain test vectors. For example, if a certain voltage exists on the gate pin, then the differences in voltages between gate and source, gate and drain, and source and drain must be within certain tolerances.

Figure 12-6. Pin Voltage Checks



Details for writing these types of checks are found under "Example: Checking Pin Voltage Conditions" and "Example: Checking Pin Voltages in Vector-Less Mode" in the *Calibre PERC User's Manual*.

Also see the *shared/examples/calibre_perc/PERC_Catalog/EOS* directory in your Calibre software tree.

Related Topics

Electrical Rule Checks

Calibre PERC Topology Rule Checks [Calibre PERC User's Manual]

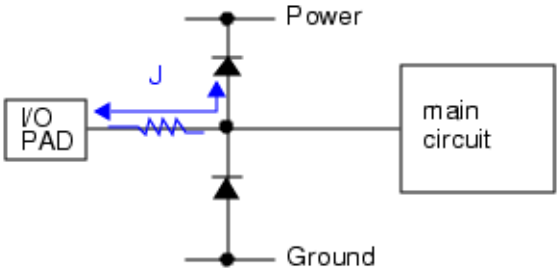
Set Up and Run Calibre PERC [Calibre PERC User's Manual]

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Current Density and Resistance Checks

In certain flows, it is desirable to calculate current density and interconnect resistivity for ESD and other types of checking. Current density and resistance is calculated between specified pins, or between ports and pins.

Figure 12-7. Current Density and Resistance



Details for writing these types of checks are found in the "Current Density Checks" and "Point-to-Point Resistance Checks" chapters of the *Calibre PERC User's Manual.*

Related Topics

Electrical Rule Checks

Calibre PERC Topology Rule Checks [Calibre PERC User's Manual]

Set Up and Run Calibre PERC [Calibre PERC User's Manual]

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Chapter 13
LVS Best Practices

There are a number of best practices to consider when using Calibre nmLVS, including rule file elements you should use, and rule file practices you should avoid.

For complete information on LVS concepts, see the *Calibre Verification User's Manual*.

For information on using Calibre Interactive - nmLVS see the *Calibre Interactive User's Manual*, and for Calibre RVE for LVS, see the *Calibre RVE User's Manual*.

Recommended LVS Rule File Statements 207
LVS Rule File Elements to Avoid 210
Hcell Recommendations for Digital Place and Route Flows 211
Efficient User-Defined Device Property Programs 213
LVS Box Statement Usage 215
Performing Only Short Isolation 218
Basic LVS Rule File Template 220

Recommended LVS Rule File Statements

LVS in Calibre is performed in two primary phases: circuit extraction and circuit comparison. Performing circuit extraction depends upon your rule file containing the appropriate layer derivations, connectivity extraction statements, text recognition statements, and device recognition statements. Hence, considerable rule file setup is required to perform circuit extraction, some of which may be performed by your foundry.

The overhead for setting up the rule file for LVS comparison is small in comparison to circuit extraction. It is a good practice to begin running Calibre nmLVS as closely as possible to the default configuration. The default settings are well chosen for most comparison situations. However, there are numerous specification statements that cover less common situations, and you can make necessary adjustments to meet your needs.

The required LVS rule file statements are discussed under "Required Statements" in the *Calibre Verification User's Manual*.

For a complete summary of LVS specification statements for use in the rule file, see "LVS Specification Statements" in the *SVRF Manual*.

This table shows recommended rule file statements that are not defaults and the circumstances for their use.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 13-1. Recommended LVS Rule File Statements

Statement	Usage
DEVice with *device_annotation_program* block.	The device annotation program block is especially useful when there are Device properties used for device characterization rather than LVS comparison. Annotation properties are calculated as DFM properties.
ERC Cell Name YES CELL SPACE XFORM	Specifies that ERC results have cell coordinates rather than global coordinates. A transformation between top-level space and cell space is provided so that you can shift back and forth in Calibre RVE.
ERC Summary Report HIER	Specifies that the ERC Summary report give results on a per-cell basis in addition to a per-rule-check basis.
Layout Base Layer	Specifies the names of device-forming layers as well as contact layers. This is a performance optimizing statement. It is important that well, substrate, via, interconnect metal, and marker or other special-purpose layers not be specified.
LVS Auto Expand Hcells PRESET [FILL]	Specifies that hcells are reported and expanded automatically during Calibre nmLVS-H circuit extraction. This is a performance optimizing statement. For LVS runs after fill has been inserted, add the FILL keyword and specify Layout Dummy Layer with the fill layers. Specifying Exclude Cell for fill cells or LVS Execute ERC NO can also be helpful.
LVS Hcell Report	Specifies to produce a detailed hcell report. The report can be particularly useful if you are unsure of your hcell selection methodology, or if you are trying to optimize an hcell list.
LVS Cpoint	Specifies a correspondence point between layout and source. This command is especially useful for corresponding net names that differ in layout and source.
LVS Ground Name	Specifies the names of ground nets in the design. This statement is used for ERC, logic gate recognition, and other LVS modules.
LVS Power Name	Specifies the names of power nets in the design. This statement is used for ERC, logic gate recognition, and other LVS modules.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 13-1. Recommended LVS Rule File Statements (cont.)

Statement	Usage
LVS Isolate Shorts {YES BY LAYER} | NO	If performing short isolation during circuit extraction, BY LAYER specifies to present the results for each layer. This allows you to highlight polygons in Calibre RVE on a per-layer basis, which is helpful in tracing shorts. There is also a BY CELL option that shows results per cell that is frequently useful. For faster LVS turnaround time, specify NO and do short isolation as a separate step. See "Performing Only Short Isolation" on page 218.
LVS Push Devices SEPARATE PROPERTIES YES	Specifies that properties not used for LVS comparison be removed from devices to facilitate pushdown after promotion. This setting is generally recommended for all flows. It is required for the pushdown separated properties (PDSP) flow that uses the Calibre Connectivity Interface in a subsequent step to LVS.
LVS Report Option BX EB EC EO ES F FX NOK R S V W	Specifies the following LVS Report contents: **BX**: List LVS Box cells. **EB**: Bad devices in circuit extraction cause an INCORRECT LVS result. **EC**: Sconnect conflicts in circuit extraction cause an INCORRECT LVS result. **EO**: Open circuits in circuit extraction cause an INCORRECT LVS result. **ES**: Short circuits in circuit extraction cause an INCORRECT LVS result. **F**: Do not report smashed MOSFETS. **FX**: Provide detailed fix suggestions for common problems. **NOK**: Report contains comparison details and statistics only for incorrect cells. Particularly useful when larger hcell lists are used. **R**: Disables reporting of ambiguity resolution points when the overall comparison result is CORRECT. **S**: Sconnect conflict details appear in the run transcript and the circuit extraction report. **V**: Reports virtual connections in the circuit extraction report. **W**: Causes LVS-H to report an error when source and layout refer to the same file.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 13-1. Recommended LVS Rule File Statements (cont.)

Statement	Usage
LVS Summary Report	Specifies that LVS generates a summary report file after a circuit comparison run. Used in conjunction with Mask SVDB Directory QUERY.
Mask SVDB Directory svdb QUERY [CCI]	The QUERY option specifies to write an LVS results database that can be loaded into Calibre RVE for debugging LVS discrepancies. Use the CCI option in a PDSP flow or if you need the Calibre Connectivity Interface of the Query Server for post-processing. For faster LVS turnaround time, you can omit Mask SVDB Directory from the rules. Specifying only the PHDB option can be useful in some cases where you need the Query Server for post-processing.
Virtual Connect Report YES	Specifies to report virtual connections when they apply.

Related Topics

LVS Rule File Elements to Avoid

Basic LVS Rule File Template

LVS Specification Statements [Standard Verification Rule Format (SVRF) Manual]

LVS Rule File Elements to Avoid

The following table shows rule file elements that should be avoided in most circumstances.

Table 13-2. LVS Rule File Elements to Avoid

Element to Avoid	Impact
Layer operations that cause device seed promotion.	Certain layer operations do not preserve layout hierarchy. When these operations are used to derive device body (seed) layers, this can cause seed promotion during connectivity extraction, which is undesirable. See "Hierarchy Preservation by Layer Operations" in the *Calibre Verification User's Manual* for details. Specifying LVS Push Devices SEPARATE PROPERTIES YES can help to mitigate seed promotion issues.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 13-2. LVS Rule File Elements to Avoid (cont.)

Element to Avoid	Impact
Shielded Connect or Sconnect statements such as these: CONNECT a b c d BY e SCONNECT f g h BY i	Connectivity statements that have more than two layers before the BY keyword cause shielding (see "Shielding" in the *Calibre Verification User's Manual*). Shielded connectivity statements can have numerous negative side-effects. It is better to rewrite shielded connectivity statements like this: CONNECT a b BY e CONNECT a c BY e CONNECT a d BY e
LVS Report Maximum ALL	The ALL setting can cause huge LVS reports that are unmanageable. The default setting is 50, which is fine for many situations. Try a setting closer to 300 if you want to see more discrepancies in the report. Discrepancies in the LVS report are typically listed with the biggest problems first, with smaller problems later. Fixing the bigger problems often resolves the smaller ones.
LVS Report Option A C	Options A and C are best avoided unless you have good reasons to use them. They can cause huge LVS reports for little gain of information. Instead, use the DETAILED INSTANCE CONNECTIONS section of the report. If option FX is used, the DETAILED ERROR ANALYSIS section is useful to review.
enclosure_vector, enclosure_parallel, enclosure_perpendicular	These are Device property computation language functions used for calculating enclosure measurements. Use the "Device Annotation Built-In Language" functions instead.

Related Topics

Recommended LVS Rule File Statements

Basic LVS Rule File Template

Efficient User-Defined Device Property Programs

Hcell Recommendations for Digital Place and Route Flows

If you are using a digital place and route (P&R) flow, the following guidelines pertain to selection of hcells.

Hcells are explicitly declared in the rule file using the Hcell specification statement or in an hcell list referenced by the -hcell command line option. Hcells may be implicitly declared for

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

LVS circuit comparison with the -automatch or -genhcells command line options, but it is frequently preferable to explicitly declare hcells.

- All of the P&R library cells should be declared as hcells.
- Cells containing devices and placed more than once, embedded memory cells, and large macro blocks all generally make good hcells. For large memory blocks, include one or two matching levels of hierarchy within the core array of the memory, if possible.
- All hcells should have their ports texted. This makes for easier debugging of LVS results. (Hcell pins are not treated as logically equivalent.)
- When multiple cell libraries are used, there is the possibility that cell names are duplicated. P&R streamout can cause cells to be renamed. If this occurs, the hcell list must be updated to include the renamed cells.

See "Hcells and Circuit Comparison" and "Hcells and Circuit Extraction" in the *Calibre Verification User's Manual* to learn more about how hcells are used, and discrepancies that can result from an non-optimized hcell list.

The LVS Hcell Report specification statement can be useful in optimizing an hcell list.

Note

If you do not have a good hcell selection methodology, then you should consider using the -genhcells command line option. If you get a CORRECT comparison using that option, then use the hcell list that this option generates in subsequent runs by specifying the list with the -hcell option, and omit -genhcells.

Related Topics

Hcells [Calibre Verification User's Manual]

Hcell and Hierarchical Analysis [Calibre Query Server Manual]

Seed Promotion and Hcells [Calibre Verification User's Manual]

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Efficient User-Defined Device Property Programs

In general, you should only extract device properties only when they are required. In LVS, that means you generally do not want to extract and compare properties used for parasitic extraction and post-layout simulation.

The Device rule file statement allows you to specify a custom device property computation program. Details of the programming language are presented under "Device Property Computation Built-In Language" in the *SVRF Manual*.

Extracting Device Properties Only When Necessary 213
Using Local Property Measurements. 214

Extracting Device Properties Only When Necessary

Many device properties are used in post-layout simulation but are not used in regular LVS debugging. Many of these properties are costly to extract in terms of LVS run time and memory consumption.

It is a good idea to avoid extracting these properties in regular LVS runs. In addition, these properties often require special auxiliary layers in Device operations. It may be important to ignore these auxiliary layers in regular LVS runs.

To achieve these goals, you can use pre-processor directives (also called conditionals).

Procedure

1. Identify the post-layout simulation properties and auxiliary layers (if any) in a Device statement. In this case, the highlighted auxiliary layers and properties are used for PEX simulation.

```
DEVICE MP pgate pgate(G) psd(S) psd(D) nw(B) <prop_layer_1>
  <prop_layer_2>

[
  PROPERTY w, l, as, ad, ps, pd
  ...
]
```

2. Change the Device statement to use conditional compilation. In this case, if the variable $PEX is set in the shell environment, then the #IFDEF blocks are compiled.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
DEVICE MP pgate pgate(G) psd(S) psd(D) nw(B)
  #IFDEF $PEX
// these layers for parasitic extraction only
  <prop_layer_1> <prop_layer_2>
  #ENDIF
[
  PROPERTY w, l
  #IFDEF $PEX
// these properties for parasitic extraction only
  , as, ad, ps, pd
  #ENDIF
  ...
]
```

Related Topics

Using Local Property Measurements

Pre-Processor Directives [Standard Verification Rule Format (SVRF) Manual]

Device Annotation Built-In Language [Standard Verification Rule Format (SVRF) Manual]

User-Defined Property Computation [Calibre Verification User's Manual]

Using Local Property Measurements

Local property measurements use seed layer declarations in Device property computation language function calls. A local measurement can use a seed layer, or a seed layer and a pin layer (the pin layer may be referenced by layer name or pin name), or a seed layer and an auxiliary layer. Using seed layers in function calls extracts the device properties in the immediate context of the device seed shapes; hence, the measurements are local.

Non-local measurements, in contrast to local measurements, do not use a seed layer at all. Such measurements are usually inefficient and should be avoided.

Procedure

1. Identify the function calls that do not use local measurements. These measurements do not reference seed layers.

```
gate = poly AND diff
pgate = gate AND pdiff

DEVICE MP pgate poly(G) psd(S) psd(D) nw(B) <aux>

[
  PROPERTY w, l, z
  z = perimeter_coincide(G, aux)
  ...
]
```

   Here, the G pin is associated with the poly layer. Poly can have large polygons that traverse the hierarchy. Using G in a function call like this can cause flattening of the poly layer. This should be avoided.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

2. Change the Device property computation function to use a device seed layer instead.

```
DEVICE MP pgate poly(G) psd(S) psd(D) nw(B) <aux>

[
  PROPERTY w, l, z
  z = perimeter_coincide(pgate, aux)
  ...
]
```

In this case, pgate is associated with a gate seed shape.

Related Topics

Extracting Device Properties Only When Necessary

LVS Best Practices

User-Defined Property Computation [Calibre Verification User's Manual]

Coding Efficiency Considerations [Calibre Verification User's Manual]

LVS Box Statement Usage

LVS Box is a rule file specification statement used to prevent the contents of cells from being compared in LVS-H verification. It is also used to exclude cell contents during circuit extraction. Typically cells that are incomplete in the layout, source, or both are specified to prevent spurious discrepancies. There are three flows: black box, gray box, and regular box. Each of these has a preferred usage model, which is important to understand when selecting a box cell flow.

Differences Between LVS Box Flows

The following table summarizes differences between the three LVS Box flows. Understanding these differences can help you select which flow best suits your needs.

Table 13-3. Differences Between LVS Box Flows

Behavior	Regular	Black	Gray
Box cells processed in isolation from rest of design	N	Y	Y
Can reduce runtime	N	Y	N
Port layers and port text required	N	Y[1]	N
Requires Mask SVDB Directory statement	N	Y	Y
Layers inside box cell hierarchy interact with outside layers	Y	N[2]	N[2]
Circuit extraction performed within box cells	Y	N	Y[3]
ERC, short isolation, and device annotation performed in box cells	Y	N	N

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Table 13-3. Differences Between LVS Box Flows (cont.)

Behavior	Regular	Black	Gray
Permits device seed promotion out of a box cell	Y	N	N
Box cell geometries preserved in SVDB for downstream flows	Y	N	Y
High-short resolution performed to box cell instance pins	Y	N	N

1. The LVS Black Box Port statement is required in the black box flow.
2. LVS Box Peek Layer overrides the default behavior.
3. By default, devices are not recognized. LVS Box GRAY DEVICES keyword overrides this.

The key difference between regular box cells (which have the longest history of support in LVS-H) and black or gray box cells is, black and gray box cells are processed in isolation from the rest of the design.

A key difference between black and gray box cells is, circuit extraction is never performed in a black box cell, while it as always performed in a gray box cell. Here are some additional considerations that are direct consequences of the behaviors listed in Table 13-3:

- Black box cells are not used for downstream flows involving parasitic extraction.
- Black box cells usually improve performance because non-pin layers are ignored.
- Black and gray box cells tend to preserve hierarchy better than regular box cells.
- Black and gray cells can cause a CORRECT result in LVS comparison in certain cases where regular box cells do not.
- Regular box cells offer the closest behavior to running LVS-H with no box cells at all.

The LVS Box LAYOUT ONLY keyword causes cells to be treated as box cells during circuit extraction only. If this statement is used, layout black and gray box cells will probably cause an INCORRECT comparison with the source because the source is not boxed, while the layout box cell will be netlisted as primitive (an exception is when LVS Box GRAY DEVICES is used, where box cells are not primitive).

Regarding LVS Netlist Box Contents and LVS Preserve Box Cells settings, the following behaviors apply:

- With default netlisting settings, regular box cells are extracted as complete subcircuits and seed promotion can occur out of the cells. Otherwise, they are extracted as primitives and no seed promotion occurs.
- Regardless of netlisting settings, black and gray box cells are extracted as primitive subcircuits with one exception. As mentioned previously, LVS Box GRAY DEVICES causes box cells to be extracted as complete subcircuits with device recognition and no seed promotion.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Black Box Cell Configuration

Black box cells are specified with the LVS Box BLACK keyword. Such cells have extra rule file requirements that other box cells do not. Specifically, the LVS Black Box Port statement must also be specified.

The LVS Black Box Port statement helps to identify black box cells. This is needed because the internal shapes of a black box cell are discarded, but the external interfaces to the cell still need to be identifiable. The LVS Black Box Port statement requires three input layers: an original (drawn) layer from which port polygons are constructed, a connectivity text layer, and a layer that externally connects to box cell ports. Following is an example.

```
LAYER m2     12 // original layer used for black box ports
LAYER txt    15 // connectivity text layer
LAYER cap    18 // capacitor marker
LAYER res    19 // resistor marker

m2wire = m2 NOT (cap OR res) // interconnect layer

CONNECT m2wire m2

// connectivity text layer must be declared in a TEXT LAYER statement
TEXT LAYER txt
ATTACH txt m2wire

LVS BOX BLACK ram // requires LVS BLACK BOX PORT statement
LVS BLACK BOX PORT m2 txt m2wire

MASK SVDB DIRECTORY svdb QUERY
```

If there are ports to a black box cell that exist below the primary level of the cell, use the LVS Black Box Port Depth statement.

LVS Gray Box Cell Configuration

Gray box cells are specified with the LVS Box GRAY keyword. Gray boxes are typically used in conjunction with downstream timing analysis tools. Gray box cells do not require any additional specification statements besides LVS Box GRAY, but there are some options that are useful in certain contexts.

Devices are not recognized by default in gray box cells. To override this, specify LVS Box GRAY DEVICES.

Like black box cells, gray box cells are processed in isolation from the rest of the layout hierarchy. This means that, by default, geometries in gray box cells do not interact with layers outside such cells. To permit layers in a gray box cell to interact with layers outside the cell, specify LVS Box Peek Layer.

Although gray box cells are processed in isolation from the rest of the design, external pin connections are determined in a separate processing step. By default, Connect or Sconnect

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

layers that form external interfaces to gray box cells form pins to such cells. To prevent a layer from forming pins in a gray box cell, use the LVS Gray Box Not Port statement.

Performing Only Short Isolation

In certain cases, it can be desirable to concentrate on performing layout short isolation over other circuit extraction tasks. With the appropriate rule file configuration and knowledge of Query Server commands, it can be faster to perform short isolation as discussed in this section.

Prerequisites

- A physical layout.
- An LVS circuit extraction rule file.
- A Calibre nmLVS/Calibre nmLVS-H license pair, a Calibre RVE license, and a layout viewer supported by Calibre RVE.

Procedure

1. Determine from your rule file the Device seed layer that defines the fewest devices in your design. Once that layer is found, declare it in an LVS Select Device By Seed Layer statement in your rules:

   ```
   LVS SELECT DEVICE BY SEED LAYER my_uncommon_device_layer
   ```

   This causes only the device instances to be extracted for the layer you have specified. (The entire Device set is not needed to perform short isolation using this method.) This usage saves runtime.

2. Declare the following statements in your rules:

   ```
   LVS ISOLATE SHORTS NO // the default

   MASK SVDB DIRECTORY svdb SI // for interactive short isolation
   ```

   These settings provide enough information to isolate specific shorts using the Query Server without isolating all possible shorts. This saves time in getting to results.

3. Run circuit extraction:

   ```
   % calibre -spice layout.sp -turbo [other options] rules |& tee log
   ```

4. Review the circuit extraction report (*.ext* file extension) to determine shorts of primary interest for fixing.

   Shorts are listed by connectivity extraction text label name (or value), location in top-level space, and layer for a given net.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
WARNING:  Short circuit - Different names on one net:
          Net Id:  1
          (1)  name  "A2"  at location  (-25,30)  on layer  10
          (2)  name  "B"  at location  (-20,30)  on layer  10
          The name "A2" was assigned to the net.
```

5. Start the Calibre Query Server Tcl shell:

```
% calibre -qs -svdb svdb

...

-------------------------------------------------------------------
-------------------------------------------------------------------
---                        CALIBRE::LVS QUERY SERVER            ---
-------------------------------------------------------------------
-------------------------------------------------------------------
Opening database "svdb/TOP_CELL.phdb/db", revision "master"
Loading hierarchy and connectivity
Loading device info
>
```

6. Execute the following command in the Tcl shell:

```
> short_db::print_shorts
```

The output gives similar information to the circuit extraction report, but is formatted differently. Here is one entry:

```
Short circuit (serial number: 0) (index: 0) (cell: TOP_CELL)
(net_id: 1) (text: A2)
    A2 at (-25,30)
    B at (-20,30)
```

Notice the shorts are organized by serial number. Locate the serial numbers of shorts of interest from step 4.

7. Using the serial number of a short from the previous step as an argument for the -sn option, execute the following command:

```
> short_db::isolate_short -sn 0
```

Successful short isolation returns a line in the transcript like this:

```
FOUND_SHORT 1 REPORT_PATH svdb/TOP_CELL.phdb/isi/TOP_CELL-default-
sn0.shorts
```

Notice the path of the short isolation database, which is an ASCII DRC-style results database. (The FOUND_SHORT return string should be handled as a Tcl array in a script.)

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

8. Open the short database in Calibre RVE from your layout viewer using whatever method you typically use (but do not simply open the Mask SVDB Directory, as this will show no shorts). Here is an example using Calibre DESIGNrev:

   ```
   % calibredrv layout.oas -rve -lvs svdb/TOP_CELL.phdb/isi/\
   TOP_CELL-default-sn0.shorts
   ```

   The database opens in Calibre RVE in interactive short isolation mode.

9. Proceed with resolving the short as discussed under "Viewing the Complete Short" and subsequent sections in the *Calibre RVE User's Manual*.

Basic LVS Rule File Template

A basic LVS rule file template is included. Modify it as necessary for your run.

```
TITLE "LVS Rules"

LAYOUT PATH "layout.gds"
LAYOUT PRIMARY "top"
LAYOUT SYSTEM GDSII

// May be used for netlist-to-netlist comparison; or use the -layout
// command line option.

    // LAYOUT PATH "layout.spi"
    // LAYOUT PRIMARY "top"
    // LAYOUT SYSTEM SPICE

SOURCE PATH "source.net"
SOURCE PRIMARY "top"
SOURCE SYSTEM SPICE

// Reporting and results output
LVS REPORT "lvs.report"
LVS REPORT MAXIMUM 50
LVS REPORT OPTION BX EB EC EO ES F FX R S V W
LVS SUMMARY REPORT "lvs.summary"
MASK SVDB DIRECTORY svdb QUERY

// Supply nets
LVS GROUND NAME "vss?"
LVS POWER NAME "vdd?"

// Hcells

// LVS AUTO EXPAND HCELLS PRESET
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
//-------Circuit Extraction section-------

// LAYER MAP statements

// LAYER statements

LAYOUT BASE LAYER poly diff contact

// TEXT LAYER statements

TEXT PRINT MAXIMUM 1000

// PORT LAYER TEXT statements

// layer derivations

// CONNECT and SCONNECT set

LVS ISOLATE SHORTS YES BY LAYER

LVS EXECUTE ERC YES

    // ERC SELECT CHECK "check names here"
    // ERC RESULTS DATABASE "erc.db"
    // ERC SUMMARY REPORT "erc.report" HIER
    // ERC CELL NAME YES CELL SPACE XFORM

    // ERC rule checks

// LVS SOFTCHK well LOWER
// LVS SOFTCHK well UPPER ALL

// DEVICE statements

//-------LVS Comparison section-------

LVS RECOGNIZE GATES ALL

// TRACE PROPERTY statements
```

Related Topics

Recommended LVS Rule File Statements

LVS Specification Statements [Standard Verification Rule Format (SVRF) Manual]

LVS Best Practices

LVS Rule File Elements to Avoid

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Chapter 14
Rule File Management: Environment Variables and Rule File Variables

Foundry-provided rule files often use conditional statements to select the portions of the rule file that are executed. The conditional statements can be controlled by environment variables or with #DEFINE statements in the rule file. Variables are also used in the rule file to set certain numerical and string parameters.

Set Environment Variables and Run Calibre From Custom Menu Item in Design Tool 223
Set Environment Variables with Calibre Interactive 225
Insert Rule File Variable and #DEFINE Statements with the Customization GUI . . . 229

Set Environment Variables and Run Calibre From Custom Menu Item in Design Tool

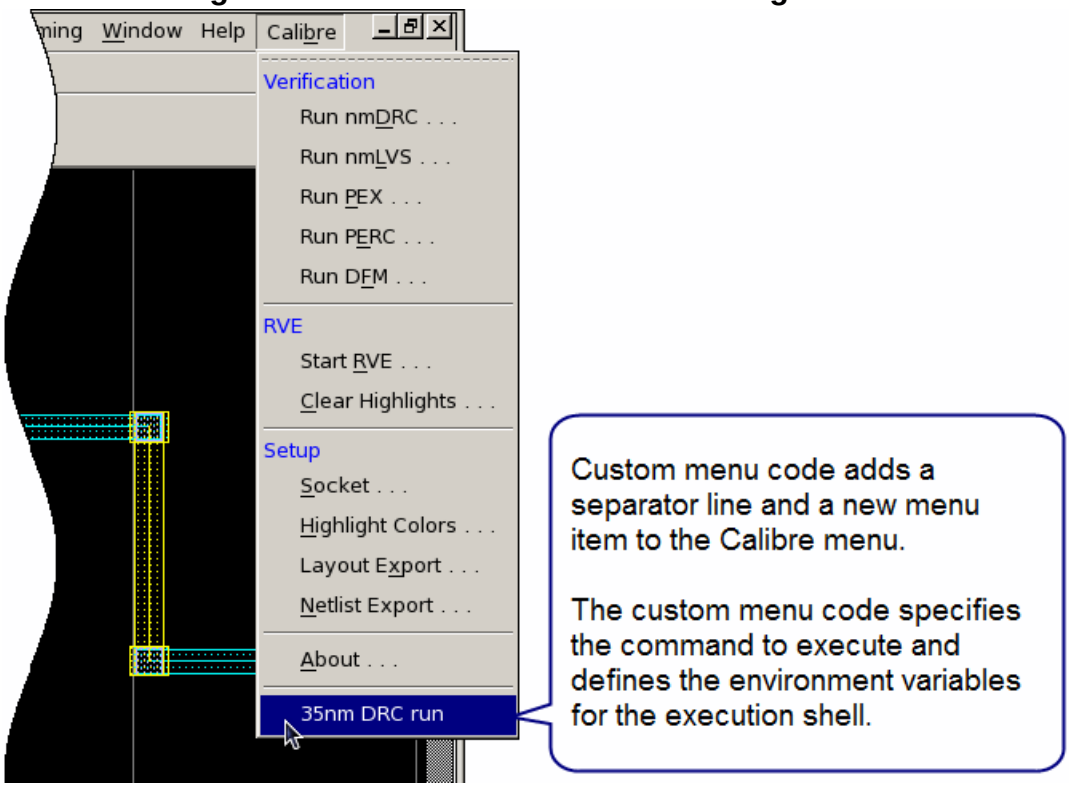
You can add custom menu items to the Calibre integration menu for supported design tools. The custom menu item can be defined to call any valid shell script or command and environment variables can be defined for the execution environment.

The custom menu feature is useful if you want to execute a command from within your design session and define the environment variables in effect for the command. Custom menus are described in the section "Custom Menu Items in Design Tools" in the *Calibre Interactive User's Manual*.

The following example adds one custom menu item and sets environment variables used during execution. Figure 14-1 shows the new menu item.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 14-1. Custom Menu Item in Design Tool



Prerequisites

- You are using one of the following supported design tools: Calibre DESIGNrev, Synopsys IC Compiler, Cadence Virtuoso, or Cadence Encounter.
- You have reviewed the section "Custom Menu Items in Design Tools" in the *Calibre Interactive User's Manual*.

Procedure

1. Create the Tcl menu customization file and name it *menu.tcl*. The following example adds a menu separator line and the menu selection "35nm DRC run" at the end of the Calibre menu.

```
# custom menu file in Tcl
#
mgc_calibre_add_menu_item -type separator

mgc_calibre_add_menu_item -label "35nm DRC run"\
    -type command -command_type CUSTOM \
    -command "\$MGC_HOME/bin/calibre -drc -hier rules" \
    -env "MGC_HOME=/calibre/sw_path" \
    -env "CALIBRE_HOME=/calibre/sw_path" \
    -env "PROCESS_VAR=35nm"
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The environment variables MGC_HOME, CALIBRE_HOME, and PROCESS_VAR are defined using the -env argument. In this example, */calibre/sw_path* is the path to the Calibre software installation.

2. In your shell environment, set the environment variable MGC_CALIBRE_VIEWER_MENU_CMDS to the name of the menu customization file. For example, using csh:

   ```
   setenv MGC_CALIBRE_VIEWER_MENU_CMDS menu.tcl
   ```

   This defines the location for the custom menu code.

3. If desired, define MGC_CALIBRE_ECHO_VIEWER_MENU_CMDS in your shell environment. Custom menu commands are echoed to the session transcript for the design tool if this environment variable is defined.

4. Start your design tool. The menu item is added as shown in Figure 14-1.

   Selecting **Calibre > 35nm DRC run** is equivalent to running the following shell script:

   ```
   setenv MGC_HOME /calibre/sw_path
   setenv CALIBRE_HOME /calibre/sw_path
   setenv PROCESS_VAR 35nm
   /calibre/sw_path/bin/calibre -drc -hier rules
   ```

Set Environment Variables with Calibre Interactive

You can use Calibre Interactive to set the environment variables that are in effect for the Calibre run and save those settings to a runset. Environment variables defined in the runset take precedence over environment variables set in the shell environment.

This method is useful for saving different run variations, since the runset saves the environment variables set within Calibre Interactive along with all the other Calibre Interactive settings. Once you have Calibre Interactive configured, you can save a runset and run Calibre Interactive in batch mode, as described in "Running Calibre Interactive in Batch Mode" in the *Calibre Interactive User's Manual*.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following example shows how to open and use the Set Environment Variables dialog box in Calibre Interactive. The rule file in this example has the following lines, where DRC_PROC is an environment variable expected to be defined in the shell:

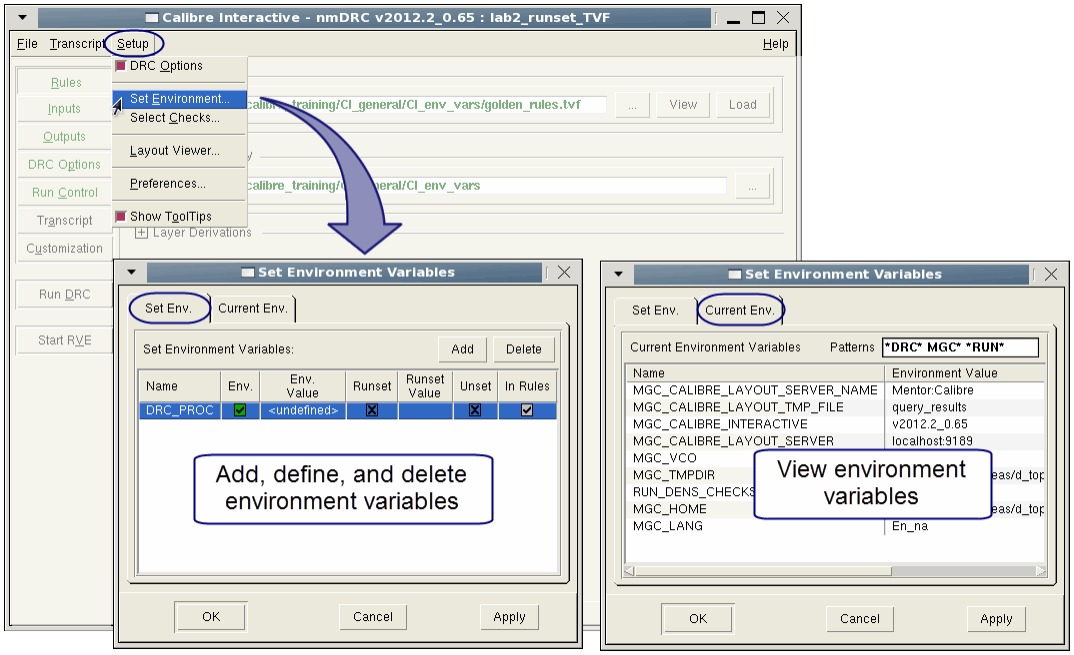
```
// SVRF rule file
// de-reference shell environment variable with $
#ifdef $DRC_PROC 7b
   LAYER top_m metal1
#else
    #ifdef $DRC_PROC 8a
        LAYER top_m metal3
    #else
          LAYER top_m metal2
    #endif
#endif
```

Prerequisites

- Calibre Interactive is open. See "Running Calibre Interactive" in the *Calibre Interactive User's Manual*.

Procedure

1. Click **Rules** on the left panel and specify the rule file.
2. (Optional) Click **Load** on the Rules pane to load settings from the rule file into the Calibre Interactive GUI fields.
3. Select **Setup** > **Set Environment** to open the Set Environment Variables dialog box.

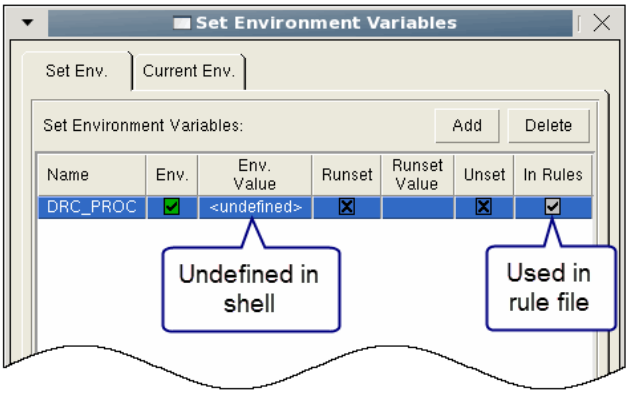


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

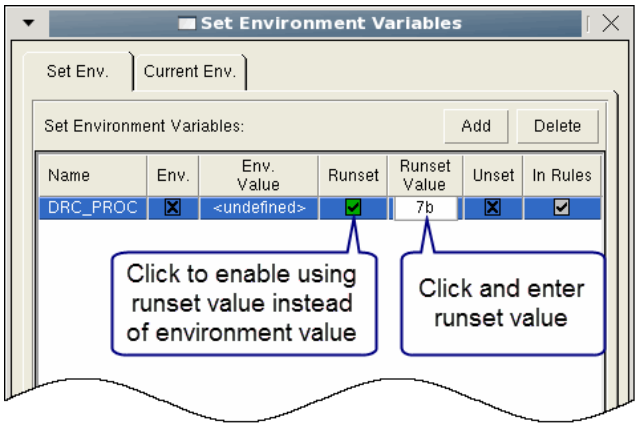
The **Set Env.** tab displays environment variables used in the rule file or defined in the runset. Environment variables used in the rule file are indicated with a check mark in the "In Rules" column.

The **Current Env.** tab displays the environment variables defined in the shell environment, subject to the filter defined in the Patterns field.

In this example, the environment variable DRC_PROC is used in the rule file, but not defined in the environment; as shown in the following figure.



4. To define the DRC_PROC environment variable in the runset, do the following:
   a. Click in the Runset Value column and enter a value.
   b. Click the checkbox in the Runset column to select the runset value.
   c. Click **OK** to dismiss the dialog box.



5. Set the rest of the Calibre Interactive options for the run.
6. Select **File** > **Save Runset** to save the runset.
7. Click **Run DRC** (or other Calibre application) to start the run.

   The runset value defined for DRC_PROC is used during the Calibre run.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Tip

Use **File** > **Load Runset** to load the runset, including the environment variable settings, into Calibre Interactive.

The runset can also be used to run Calibre Interactive in batch mode, without opening the GUI.

Related Topics

Adding a Runset Environment Variable Example [Calibre Interactive User's Manual]

Editing a Runset or Rule File Environment Variable Example [Calibre Interactive User's Manual]

Changing the Value of a Shell Environment Variable Example [Calibre Interactive User's Manual]

Running Calibre Interactive in Batch Mode [Calibre Interactive User's Manual]

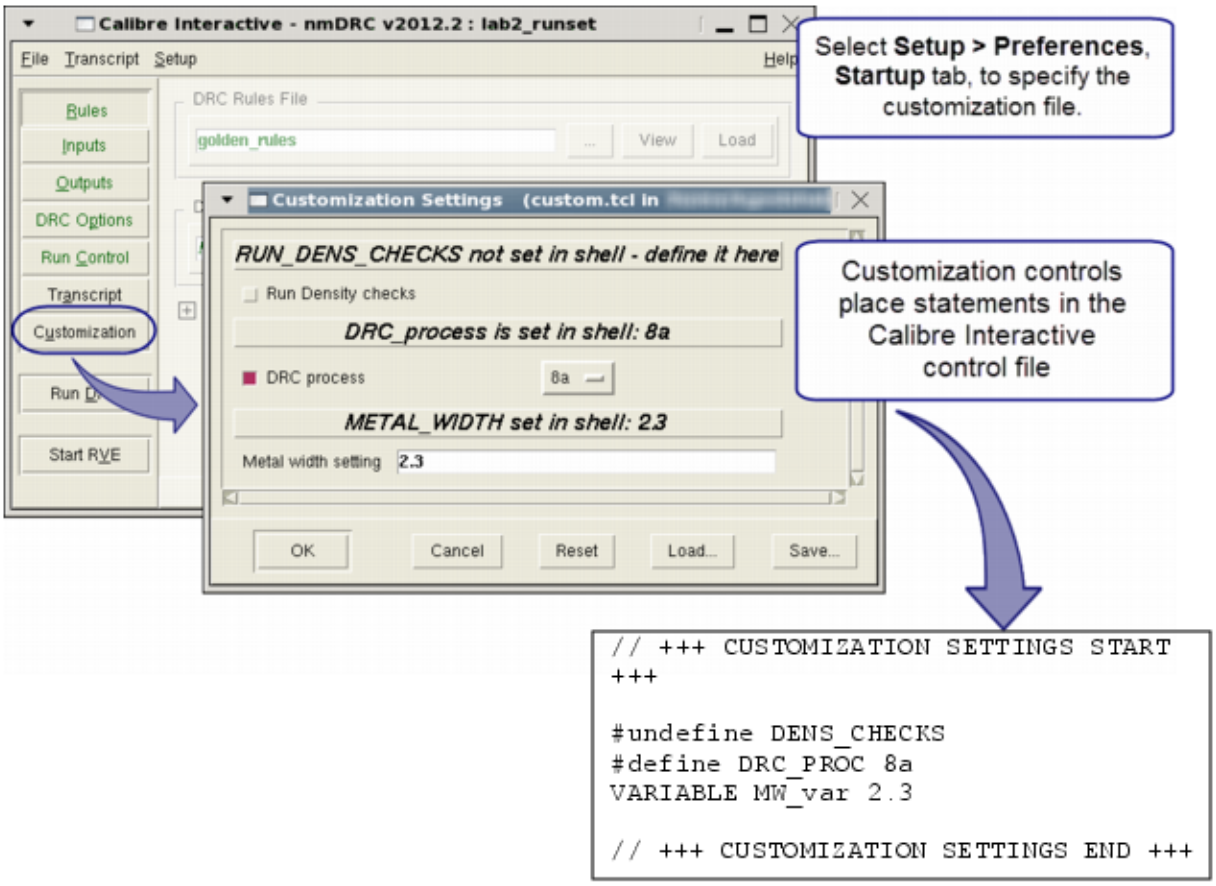
Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Insert Rule File Variable and #DEFINE Statements with the Customization GUI

You can use the Customization GUI in Calibre Interactive to place #DEFINE and #UNDEFINE pre-processor directives and variable definitions in the rule file. The customization GUI is a dialog box with user-coded controls which place statements in the Calibre Interactive control file.

The customization GUI is coded using the Tcl language, as described in "Customization Files" in the *Calibre Interactive User's Manual*. The following figure shows a customization GUI based on the examples in the following section.

Figure 14-2. Customization GUI in Calibre Interactive



The Customization GUI can be used with both interactive and batch execution.

Since the customization file is written in Tcl, it can read environment variables set in the shell environment and use them in coding the customization controls.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

These examples are included:

"Insert a Boolean #DEFINE Based on an Environment Variable" on page 230

"Insert a #DEFINE with a Value Based on an Environment Variable" on page 232

"Setting a Rule File Variable with the Customization GUI" on page 234

Note

When using the Customization GUI to insert #DEFINE or #UNDEFINE statements into the Calibre Interactive control file, be sure that the rule file does not contain any #DEFINE or #UNDEFINE statements for the same variable, since later statements take precedence over those in the control file.

See the topic "Methods to Specify a Customization File" in the *Calibre Interactive User's Manual* for information on loading a customization file. See the other topics in the chapter "Customization Files" for details about the customization language and commands.

Insert a Boolean #DEFINE Based on an Environment Variable

This example demonstrates using the Calibre Interactive Customization GUI to place a #DEFINE pre-processor directive in the control file. A shell environment value is read to determine the initial value of the #DEFINE directive.

Suppose an SVRF rule file uses the following code with a pre-processor directive based on DENS_CHECKS:

```
// SVRF rule file
#ifdef DENS_CHECKS
metal1_density {
    @ density metal1 greater than 50%, or 25%
    DENSITY metal1 < 0.5 WINDOW 50 RDB lab2_m1_density.rdb
}
#endif
```

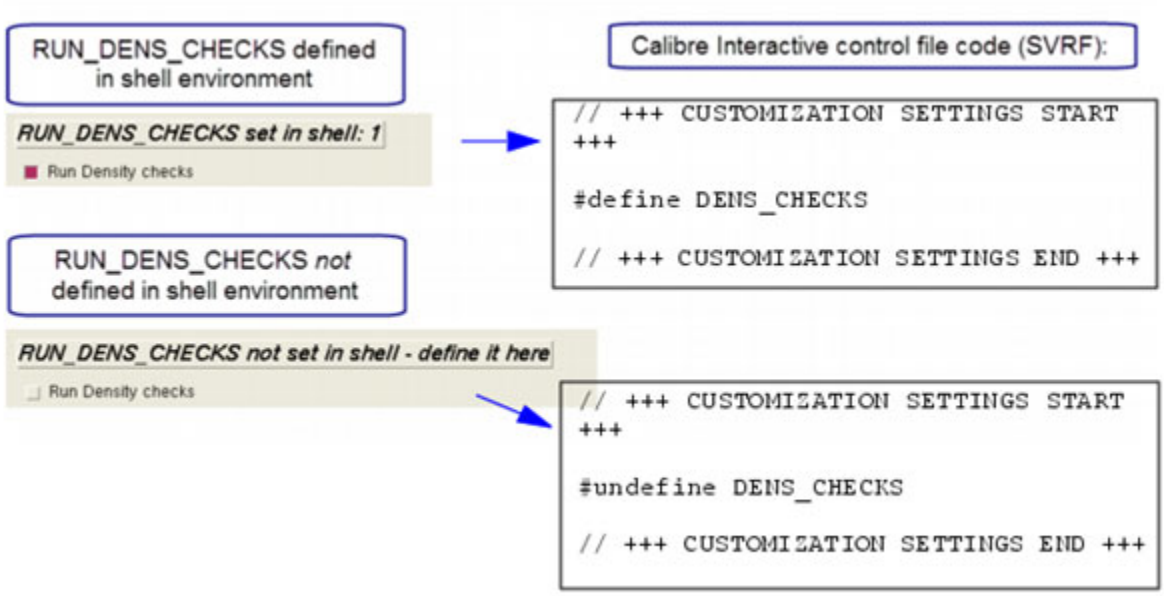
In this example, DENS_CHECKS is not expected to be defined in the shell environment, since it is not preceded with a $ character. The following customization GUI code determines if the shell environment variable RUN_DENS_CHECKS is defined, then uses that information to place a #DEFINE or #UNDEFINE directive for DENS_CHECKS in the Calibre Interactive control file.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
# customization file

# RUN_DENS_CHECKS is a Boolean that may be set in the shell environment
if {[info exists ::env(RUN_DENS_CHECKS)]} {
     CUSTOM::LABEL  -tool DRC \
       -prompt "RUN_DENS_CHECKS set in shell: $::env(RUN_DENS_CHECKS)"
   # create Boolean checkbox which defines or un-defines DENS_CHECKS
   # setting initialized to the RUN_DENS_CHECKS env var
   # command defines or un-defines DENS_CHECKS
  CUSTOM::DEFINE -name "DENS_CHECKS" -prompt "Run Density checks" \
    -tool DRC -select $::env(RUN_DENS_CHECKS)
} else {
  CUSTOM::LABEL -tool DRC \
    -prompt "RUN_DENS_CHECKS not set in shell - define it here"
  # create Boolean checkbox which defines or un-defines DENS_CHECKS
  CUSTOM::DEFINE -name "DENS_CHECKS" -select 0 \
    -prompt "Run Density checks" -tool DRC
}
```

This customization file produces the result shown in the following figure. A view of the complete Customization Settings dialog box is shown in Figure 14-2 in "Insert Rule File Variable and #DEFINE Statements with the Customization GUI" on page 229.

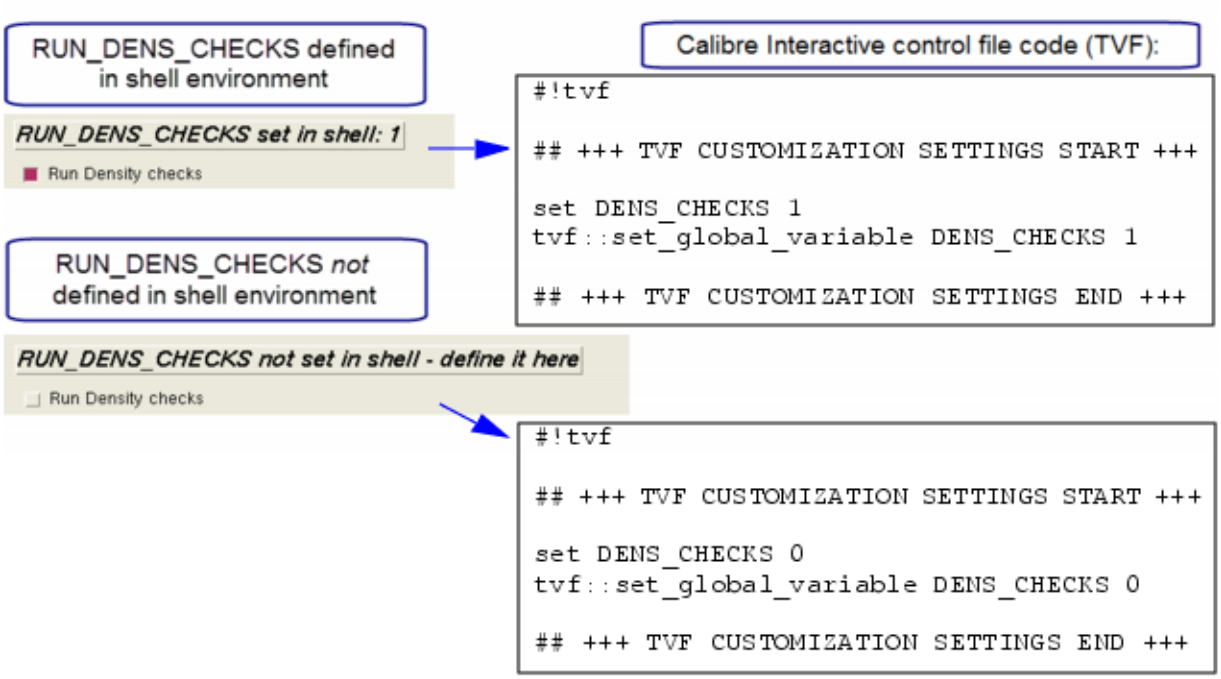
Figure 14-3. Customization GUI Results for Boolean #DEFINE



If you are using a TVF rule file, you add the "-tvf" option to the CUSTOM::DEFINE statements, which adds the code shown in the next figure to the Calibre Interactive control file for this example.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 14-4. Customization GUI Results for Boolean Define in TVF Rule File



Insert a #DEFINE with a Value Based on an Environment Variable

This example demonstrates how to read the value of a shell environment variable using the Calibre Interactive Customization GUI and insert a #DEFINE with that value into the Calibre Interactive control file.

Suppose the SVRF rule file has this code:

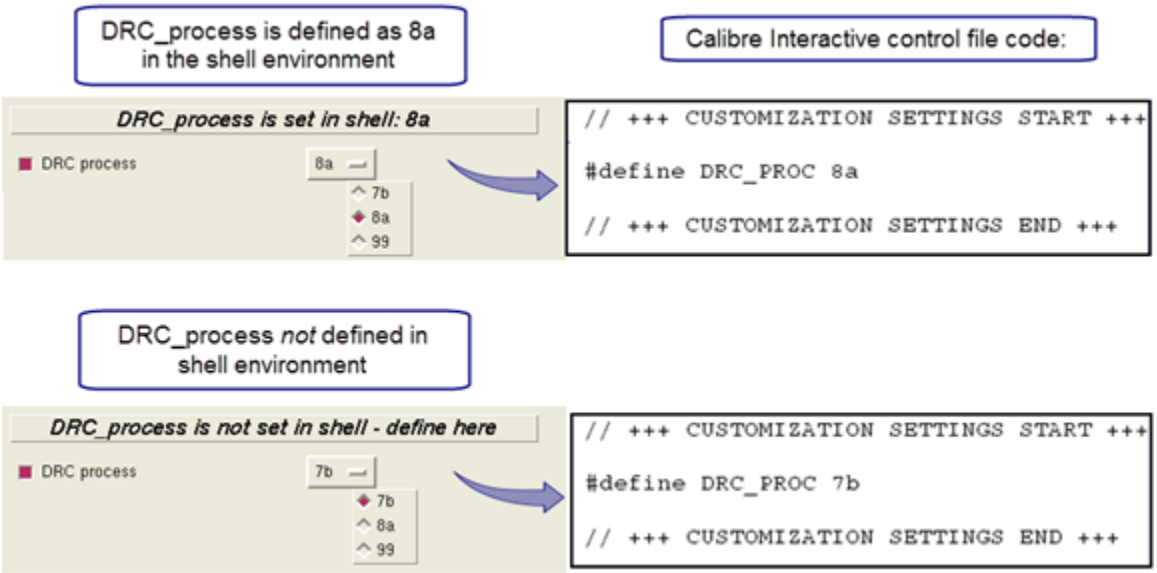
```
// SVRF rule file
// define layer top_m based on process set by DRC_PROC
#ifdef DRC_PROC 7b
   LAYER top_m metal1
#else
    #ifdef DRC_PROC 8a
        LAYER top_m metal3
    #else
        LAYER top_m metal2
    #endif
#endif
```

where the definition of the layer top_m depends on the value of DRC_PROC. DRC_PROC is assumed to be defined in the rule file. Suppose you want to set the environment variable DRC_process in the shell environment to control DRC_PROC in the rule file. The following customization file accomplishes this.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

```
# customization file in Tcl
#
if {[info exists ::env(DRC_process)]} {
  CUSTOM::LABEL -tool DRC \
    -prompt "DRC_process is set in shell: $::env(DRC_process)"
  CUSTOM::DEFINE -name "DRC_PROC"  -prompt "DRC process" \
    -choices  [list [list "shell value" $::env(DRC_process)] \
              [list "7b" 7b] [list "8a" 8a]  [list "other" 99]] \
    -initval $::env(DRC_process) -tool DRC
} else {
  CUSTOM::LABEL -tool DRC \
    -prompt "DRC_process is not set in shell - define here"
  # this command defines DRC_PROC to the selection from the choices list
  CUSTOM::DEFINE -name "DRC_PROC" -prompt "DRC process" -tool DRC \
    -choices [list 7b 8a 99]
}
```

The following figure shows the results for the preceding example; a view of the complete Customization Settings dialog box is shown in Figure 14-2 in "Insert Rule File Variable and #DEFINE Statements with the Customization GUI" on page 229.



If you are using a TVF rule file, add the "-tvf" option to the CUSTOM::DEFINE statements, which places a Tcl variable in the Calibre Interactive control file instead of a #DEFINE pre-processor directive. Figure 14-4 shows results when the -tvf option is used (from the example "Insert a Boolean #DEFINE Based on an Environment Variable" on page 230).

Note: If you want to read DRC_process from the shell environment and not allow changes to the value from the Customization GUI, you can code the control with only one choice in the -choices argument. It is still necessary to include the -choices argument if you want to assign a value to with the #DEFINE directive; without the -choices argument the #DEFINE directive is

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

used without a value (as in the example "Insert a Boolean #DEFINE Based on an Environment Variable" on page 230). For example:

```
# this code reads DRC_process from the shell and sets DRC_PROCtest
# without allowing change from the customization gui
if {[info exists ::env(DRC_process)]} {
  CUSTOM::LABEL -tool DRC \
    -prompt "DRC_process is set in shell: $::env(DRC_process)"
  CUSTOM::DEFINE -name "DRC_PROCtest"  \
    -prompt "DRC process =$::env(DRC_process)" \
    -choices $::env(DRC_process) -tool DRC
}
```

Setting a Rule File Variable with the Customization GUI

You can use the customization GUI to insert a Variable statement in the Calibre Interactive control file. The following example reads the environment variable METAL_WIDTH from the shell and uses the value in a Variable statement.

Suppose the SVRF rule file has this code, where MW_var is an SVRF variable used to specify the width constraint.

```
// width constraint set by svrf variable MW_var
min_poly_width {
 @ Minimum poly width = ^MW_var
 int poly < MW_var opposite
}
```

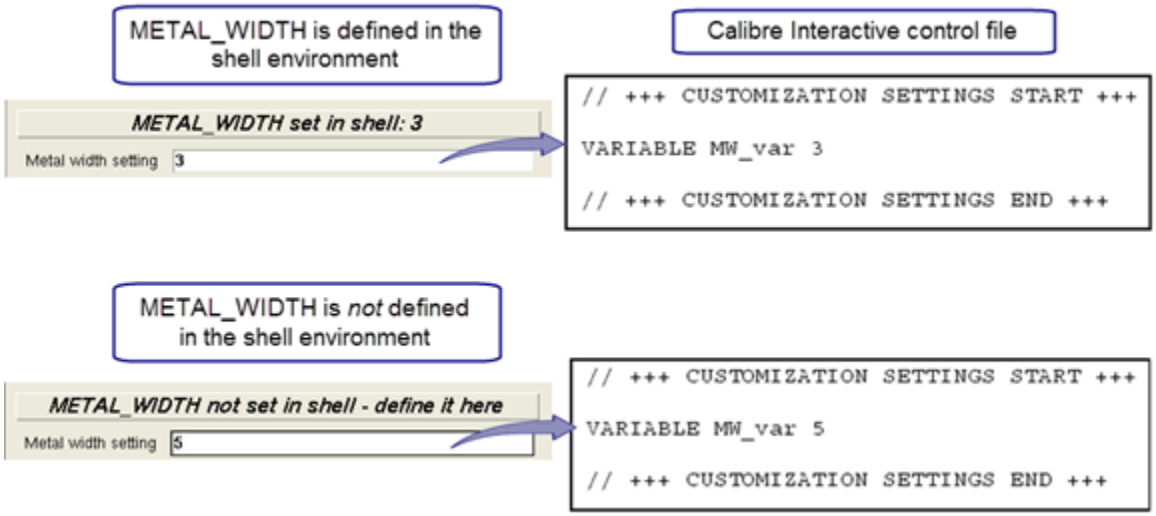
Then suppose that you want to set the value of MW_var using the environment variable METAL_WIDTH. The following customization file does that.

```
if {[info exists ::env(METAL_WIDTH)]} {
  CUSTOM::LABEL  -tool DRC \
   -prompt "METAL_WIDTH set in shell: $::env(METAL_WIDTH)"
  CUSTOM::VARIABLE  -prompt "Metal width setting" \
    -name "MW_var" -initval $::env(METAL_WIDTH) -tool DRC
} else {
  CUSTOM::LABEL -tool DRC \
   -prompt "METAL_WIDTH not set in shell - define it here"
  CUSTOM::VARIABLE -name "MW_var"  \
   -prompt "Metal width setting" -tool DRC
}
```

The following figure shows the results for the preceding example, where a Variable statement is placed in the control file; a view of the complete Customization Settings dialog box is shown in

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Figure 14-2 in "Insert Rule File Variable and #DEFINE Statements with the Customization GUI" on page 229.



If you are using a TVF rule file, add the "-tvf" option to the CUSTOM::VARIABLE statements, which places a Tcl variable in the Calibre Interactive control file.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Chapter 15
Pre- and Post-Processing for Calibre Runs

Some form of pre- or post-processing is often performed with a Calibre run. For example, pre-processing can prepare a layout or netlist for verification or construct an input file. Post-processing can prepare or analyze report files. There are several methods for automatically executing a pre- or post-processing procedure or script with your Calibre run. The method you choose depends on your design environment and the execution environment for the pre- or post-processing procedure.

Table 15-1. Pre- and Post-Processing Methods with Calibre Verification

Method	Execution environment	Pre- or post-processing	In Calibre Interactive runset?
External trigger in Calibre Interactive These run in the design tool before and after Calibre Interactive execution. External triggers are typically defined in the initialization code for the design tool. See "External Triggers in Calibre Interactive"	Design tool[1]	Both	No
Internal trigger in Calibre Interactive Internal triggers are part of Calibre Interactive and execute before and after the Calibre run. See "Internal Triggers in Calibre Interactive"	Shell or Cadence Virtuoso	Both	Yes
Custom menu command You can add custom menu items in supported design tools and tie the menu item to one or more commands. In Cadence Virtuoso, you have the option of defining a Skill function to run before the main (shell) command. See "Custom Menu Items in Design Tools"	Shell or Cadence Virtuoso	Pre-processing in Cadence Virtuoso	No

1. Calibre DESIGNrev, Cadence Virtuoso, and Synopsys IC Compiler

The following examples demonstrate pre- and post-processing using Calibre tools.

Creating a One-Click Custom Menu Command With SKILL Pre-Processing 238
Processing a Netlist Before the Calibre Run 240

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Creating a One-Click Custom Menu Command With SKILL Pre-Processing

You can define a custom menu command for Cadence Virtuoso to first call a SKILL function, then execute one or more commands. This example demonstrates how to create a custom menu item that calls a SKILL procedure, runs Calibre Interactive in batch mode, then executes a shell command for post-processing.

Here are the desired actions for this scenario:

- Run a SKILL procedure to create a needed include file for the Calibre run and set an environment variable pointing to the include file.
- Run batch Calibre Interactive with a runset that specifies the include file using an environment variable. Calibre Interactive automatically opens Calibre RVE after the run completes.
- Run a post-processing shell script to create a report.

This can all be done in one click from the Cadence Virtuoso menu using custom menu commands. The new Calibre menu selection executes all three of the steps listed above. The following steps are needed to accomplish this:

1. Create the SKILL pre-processing procedure.
2. Create and specify the Tcl custom menu file.
3. Set up a runset for Calibre Interactive batch operation.
4. Start Cadence Virtuoso and call the new custom menu command.

The following procedure describes these steps in detail, including code examples.

Note

Pre-processing in the design environment from a custom menu command is only available for Cadence Virtuoso.

Procedure

1. Create the SKILL pre-processing function. Here is an example:

```
procedure(set_include(run_type)
    ;; create file
    p = outfile(strcat(run_type ".include"))
    ;; commands to create include file
    close(p)
    ;; set the environment variable pointing to the file
    setShellEnvVar(strcat("INCLUDE_FILE=./" run_type ".include"))
); end set_include
```


Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

For example, if set_include("pex") is called, then the file *pex.include* is created and the environment variable INCLUDE_FILE is set to *./pex.include*.

2. Create the customization menu file and save it as *menu_custom.tcl*. Here is an example customization menu file:

```
# for Cadence Virtuoso
# delete unwanted menu items
mgc_calibre_delete_menu_item -label "Run DFM"
mgc_calibre_delete_menu_item -label "Run PERC"

# create command
# must escape the quotes around the argument for set_include
mgc_calibre_add_menu_item -type command -label "pex_run" \
    -command_type PEX \
    -command "calibre -gui -pex -runset pex.runset -batch; \
    my_report.sh" \
    -code "set_include(\"pex\")"
```

   The mgc_calibre_add_menu_item command adds the menu item "pex_run" to the Calibre menu. The following actions are tied to the menu item and occur in this order:

   a. Executes the SKILL procedure set_include("pex"). This is specified with the -code option.

   b. Executes "calibre -gui -pex -runset pex.runset -batch".

   c. Executes "my_report.sh". This command is separated from the first command with a semicolon.

3. Specify the custom menu file with an environment variable and specify that commands are echoed:

```
% setenv MGC_CALIBRE_VIEWER_MENU_CMDS menu_custom.tcl
% setenv MGC_CALIBRE_ECHO_VIEWER_MENU_CMDS 1
```

4. Set up your Calibre Interactive run for batch operation and save a runset:

   a. Make all desired run settings for your application.

   b. (Optional) Disable **Setup** > **Preferences** > **Misc**, option "Ignore layout and source export in batch mode". This allows export to occur in batch mode if the design is open in a connected design tool.

   c. Enable **Setup** > **Preferences** > **Misc**, option "Start RVE after a batch run" (runset option: cmnStartRVEAfterBatch).

   d. Choose **Setup** > **PEX Options**, then click the **Include** tab. Enter $INCLUDE_FILE in the area for "Include Rule Files".

      The SKILL pre-processing procedure created in Step 1 creates the include file and sets the environment variable.

   e. Select **File** > **Save Runset** to save the runset.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

5. Start Cadence Virtuoso and open a layout. Make sure the SKILL function set_include() is loaded in the environment.

   The new menu item "**pex_run**" is at the bottom of the Calibre menu.

6. Click the **pex_run** menu item.

Results

The following actions occur in order when you select the new **pex_run** menu item:

- The SKILL procedure set_include("pex") executes. It creates the file *pex.include* and sets the environment variable INCLUDE_FILE to *./pex.include*.
- Calibre Interactive runs in batch mode. Calibre RVE is opened when the run completes.
- The shell script "my_report.sh" executes.

Related Topics

Custom Menu Items in Design Tools [Calibre Interactive User's Manual]

Running Calibre Interactive in Batch Mode [Calibre Interactive User's Manual]

Processing a Netlist Before the Calibre Run

You can use Calibre Interactive to specify a trigger function that runs just before the Calibre run starts. This example defines a trigger function to process the source netlist. The Calibre Interactive internal trigger runs after source and layout export completes and before the Calibre nmLVS run starts.

Here are the desired actions for this Calibre nmLVS scenario:

- Export the source and layout from the design tool.
- Run a script to process the source netlist.
- Run Calibre nmLVS.
- Start Calibre RVE.

You can do this using Calibre Interactive with an internal trigger. The name of the source netlist is given to the trigger function as a trigger parameter. The following steps are needed to accomplish this:

1. Create the shell script to process the source netlist.
2. Specify the shell script on the **Setup** > **Preferences** > **Triggers** tab in Calibre Interactive.
3. Set up the Calibre Interactive runset.
4. Open your design and start the Calibre Interactive run.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

The following procedure describes these steps in detail, including code examples.

Procedure

1. Create the shell script for editing the netlist file. This example takes the netlist name as an argument and creates a copy.

```
#!/bin/sh

echo "executing internal trigger, copy source netlist"
# expects source netlist filename as argument
echo "source netlist is: $1"
cp $1 $1.copy
```

   Save the file and name it *copy_netlist.sh*.

2. Open Calibre Interactive nmLVS.

3. Do the following to specify the shell script *copy_netlist.sh* as a trigger function.

   a. Select **Setup** > **Preferences**, then choose the **Triggers** tab.

   b. In the **Pre-Execution** tab, click the **Add Triggers** button ( ) to add a new trigger function.

   c. Enter the command and any arguments in the Trigger column.

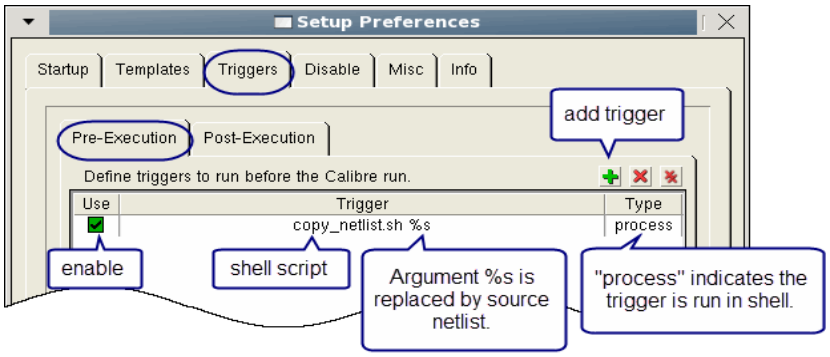
```
copy_netlist.sh %s
```

   In this case the argument is the trigger parameter %s, which is replaced with the path to the source netlist at runtime. Available trigger parameters are listed on the **Triggers** tab.

   d. Click the checkbox in the Use column to enable the trigger.

   e. Verify that “process” is entered in the Type column. This specifies that the trigger function is run in the shell environment.

   The **Setup** > **Preferences** > **Triggers** tab should display as follows:



Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Tip

For Cadence Virtuoso, you can specify “layout” or “schematic” in the Type column to run a Skill procedure in the layout or schematic viewer.

f. Click **OK** to close the Setup Preferences dialog box.

4. Click **Inputs** on the left panel and do the following to specify layout and source export from the design tool:
   a. Select the **Layout** tab and enable “Export from layout viewer”.
   b. Select the Netlist tab and enable “Export from schematic viewer”.
5. Click **Outputs** on the left panel, and enable “Start RVE after LVS finishes”.
6. Select other Calibre Interactive settings for the run.
7. Select **File > Save Runset** to save the runset.

   You are now ready to start the Calibre Interactive run.
8. Open the layout and source in the design tool.
9. Start Calibre Interactive nmLVS and load the runset you saved in Step 7.
10. Click **Run LVS** to start the run.

Results

The following actions occur in order when you click **Run LVS**:

- The layout and source are exported from the design tool.
- The shell script *copy_netlist.sh* executes.
- The Calibre nmLVS run starts.
- Calibre RVE opens.

Related Topics

Internal Triggers in Calibre Interactive [Calibre Interactive User's Manual]

Running Calibre Interactive [Calibre Interactive User's Manual]

Interfacing Calibre Interactive to Layout and Schematic Viewers [Calibre Interactive User's Manual]

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Index

— Symbols —

[], 14
{}, 14
|, 14

— B —

Bold words, 13

— C —

Calibre database, 140
Check for empty layer, 136
Courier font, 13
Current density checks, 205
Customization GUI
    insert #DEFINE, 230, 232
    insert Variable statement, 234
    overview, 229

— D —

Database grid, 139
dbu (database unit), 139
Density, 111
    color map, 116
    combining windows to reduce data, 123
    DRC RVE setup for using RDB data, 112
    histogram, 114
    inside of a layer, 119, 123
    INSIDE OF LAYER options, 129
    outside of a cell or marker layer, 117
    results in Calibre RVE, 111
    solution types, 117
Derive empty layer, 135
Device count, 195
    flat, 195, 197
    total, 196
Device properties
    efficiency, 213, 214
Discrete coordinate system, 139
Double pipes, 14
DRC
    density, 117
    equation-based, 15
    layout magnification, 139
DRC RVE HTML reports, see HTML reports in Calibre RVE for DRC

— E —

Empty layers, 135
Environment variables
    set from Calibre Interactive, 226
    set from custom menu, 223
Equation-based DRC checks
    contact balance, 19
    current crowding - contact placement, 22
    current crowding - contact run length, 21
    device orientation, 29
    equal gate pitch per active area, 62, 81
    gate width dependent via count, 32
    maximum via coverage for metal overlap regions, 25
ERC checks, 199
ESD checks, 199

— G —

Gate protection, 204
Ground domain separation, 202

— H —

Hcells, 211
Heavy font, 13
HTML reports in Calibre RVE for DRC
    Calibre Auto-Waivers example, 182
    Calibre DRC example, 182
    Calibre LFD example, 191
    Calibre PERC example, 189
    overview, 181

— I —

I/O pad protection, 203
Italic font, 13

— L —

Layout database, 140
    scaling, 144
Layout magnification, 139, 149
    specification statements, 141
Layout rotation, 153
LVS
    best practices, 207
    hcells, 211
    recommended statements, 208
    rule file template, 213
    things to avoid, 210

— M —

Magnification factors, 144, 147
Measurement precision, 141, 148
Minimum keyword, 14
MOSFET bulk pin connections, 200

— P —

Parentheses, 14
PERC usage, 199
Physical precision, 141
Pipes, 14
Power domain checks, 201
Precision, 141

— Q —

Quotation marks, 14

— R —

Resistance checks, 205
Rule file
    LVS, 213
    precision, 141
RVE batch reports, see HTML reports in
        Calibre RVE for DRC

— S —

Scaling factors, 144
Slanted words, 13
Square parentheses, 14
Syntax conventions, 13

— U —

Underlined words, 13

— V —

Voltage checks (pins), 205

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.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.

Note - Viewing PDF files within a web browser causes some links not to function. Use HTML for full navigation.