Virtuoso Multi-Patterning Technology User Guide

Product Version ICADVM20.1
October 2020

Contents

Preface 9

Scope 9
Licensing Requirements 10
Related Documentation 10
  What's New and KPNS 10
  Installation, Environment, and Infrastructure 10
  Technology Information 10
  Virtuoso Tools 10
Additional Learning Resources 11
  Video Library 11
  Virtuoso Videos Book 11
  Rapid Adoption Kits 11
  Help and Support Facilities 12
Customer Support 12
Feedback about Documentation 13
Understanding Cadence SKILL 14
  Using SKILL Code Examples 14
  Sample SKILL Code 14
  Accessing API Help 15
Typographic and Syntax Conventions 16
Identifiers Used to Denote Data Types 17

1
Introduction 19

Multi-Patterning Lithography 20
  Multi-Patterning Technology 20
Introducing Virtuoso Multi-Patterning Technology 22

2
Getting Started 25

Prerequisites 26
  Specifying the Number of Masks Per Layer 26
  Specifying Same-Mask and Diff-Mask Spacing Constraints 27
  Specifying Coloring Purposes 28
  Example of ASCII Technology File Data for MPT 29
Customizing Your Environment 33
  Enabling the Multiple Patterning Color Engine 33
  Using the Multiple Patterning Options Form 35
  Customizing Displayed Coloring 39

3
Net-Based Pre-Coloring Flow 43

Overview 44
Requirements 44
Identifying the Critical Nets 44
Assigning the Pre-Coloring Constraints 45
Viewing Pre-Colored Nets in Layout 45
  Updating Layout Constraints 45
  Same Mask Pre-Coloring 46
Streaming Out Pre-Colored Nets 46

4
Coloring in Layout 49

Interactive Coloring in Layout 50
  Color Representations 51
  Creating Colored Shapes 52
  Controlling the Visibility and Selectability of Colored Data 52
  Inspecting Colored Data 53
  Coloring Existing Shapes 56
  Color Locking 56
  Color Shifting 61
  Pre-defined Setup Driven MPT Flows 64

Using the Multiple Patterning Toolbar 65
- Viewing the Last Operation Status and Probing the Color Source 90
- Probing Color Source 91

Using the Multiple Patterning Color Engine 94
- Hierarchical Coloring of Connected Shapes 94
- Automatic Color Shifting 95
- Color Shifting for Cells 96

Track-Based Coloring 98
- Coloring Shapes that Overlap Colored Tracks 98
- Creating Colored Tracks Using Width Spacing Patterns 99
- Creating Colored Tracks Using Track Patterns 100
- Displaying Track Patterns in Layout 103
- Assigning Track Patterns to Nets 103

Migrating from the Multiple Patterning Assistant 110

Saving and Restoring Data 111

Limitations 112

5

Checking and Fixing Multiple Patterning Violations in Layout

113

Types of Color Checks 114
- Color Rule Violations 114
- Color Shorts 114
- Uncolored Shapes 115
- Colorability 115

Checking for Multiple Patterning Violations 117
- Using Color-Aware DRD Edit 118
- Using Pegasus Interactive 122

Using the Annotation Browser to View Multiple Patterning Violations 123

Fixing Multiple Patterning Violations 127
- Using Stitch for Multiple Patterning Violations 128

Verifying the Consistency of Color Assignments 132
- Virtuoso Coloring Check 132
- LVS Coloring Check 134

6
Importing and Exporting Data 137
Using XStream 138
Exporting Stream Files with Coloring 138
Importing Stream Files with Coloring 141
Using Enhanced Layer Map Files for Coloring 143
Types of Color Representations 143
Handling Stitches 144
Warning Messages from XStream 144
Layer-Purpose Pair to Color Data 145
Merging Layer-Purpose Pairs on Two Layers (2-to-1) 145
Merging Multiple Layer-Purposes on One Layer (1-to-1) 145
Transforming Shapes on Specified Purposes to Colored Blockages 146
Color Data Migration 147
From/To IC6.1.x 147
To ICADV12.2 LA4 or above 147
To ICADV12.2 Base Release or ICADV12.2 ISR1 148
mptScan 150

7
Fully Colored Backannotation Flow 155
Overview 156
Creating the Layer Map File 157
Translating Data to a Stream File 157
Assigning Colors to Unlocked and Uncolored Shapes Using a Third-Party Color Engine 157
Merging Locked Shapes with Shapes Colored by the Third-Party Color Engine . . . 157
Creating a Stream File with All Colors Locked 157
Enhancements in the Layer Map File Format 158
Using colorAnnotate 161

A
MPT Environment Variables 165
Environment Variable Functions 166

Checking the Value of an Environment Variable 166
Setting the Value of an Environment Variable 166
List of Virtuoso MPT Environment Variables 168
allowLockShiftOverride 171
autoPropagateLock 172
checkHCLOverUnlock 173
colorCDFCheck 174
colorCDFParamPrefix 175
colorConstFileColorAName 176
colorConstFileColorBName 177
colorConstFileName 178
coloredPurposeTypes 179
coloringEngineEnabled 180
coloringFilterSize 181
colorShiftingLayers 182
complianceCheckerLimit 183
complianceCheckerReport 184
defaultColoringMethod 185
deleteConnectedShapes 186
displayMaskColor 187
displayMaskColorExcludeViaLayers 188
displayMaskColorMode 189
displaySystemColor 190
dontColorPCells 191
drawSurroundingOn 192
enableHCLCreation 193
enableHCLCreationOnPcells 194
enableMarkersToMaskColors 195
enforceWSPColor 196
explicitColoredPurposes 197
extractorStopLevel 198
forcePcellRecolorOnEval 199
forceRandomOnLayerType 200
globalColorShiftingPolicy 201
globalColorShiftingPolicyForPcells 202
hideStitchingTools 203

lockAllHCLPolicy 204
mergeColoredPacket 205
mptConstraintGroup 206
onlyCheckActiveWSP 207
overrideLockOnConnectedShapes 208
propagateLocksToConnectedShapes 209
propagateAnySameMaskState 210
pvsDeckFile 211
pvsLayerMapFile 212
reColorGUIScope 213
reColorReadOnlyCellView 214
shiftCutColorFromToolbarButton 215
showCDFchecks 216
trackColoringOnlySnappedShapes 217
unclusteredShapeColor 218
updateColorOnActivate 219
copyMPAttributes 220

B
MPT Functions 221

Virtuoso MPT SKILL Functions 222
Virtuoso SKILL Functions for MPT 228

C
MPT Forms Reference 233

Export Color Constraint File 234
Multiple Patterning Options 235
Markers To Mask Colors 237

Preface

Virtuoso® Multi-Patterning Technology (Virtuoso MPT) helps you to visualize how an advanced node layout can be decomposed to mask colors.

This user guide is aimed at developers and designers of integrated circuits and assumes that you are familiar with:

- The Virtuoso design environment and application infrastructure mechanisms designed to support consistent operations between all Cadence® tools.
- The applications used to design and develop integrated circuits in the Virtuoso design environment, notably, the Virtuoso Layout Suite, and Virtuoso Schematic Editor.
- The Virtuoso design environment technology file.

This preface contains the following topics:

- Scope
- Licensing Requirements
- Related Documentation
- Additional Learning Resources
- Customer Support
- Feedback about Documentation
- Understanding Cadence SKILL
- Typographic and Syntax Conventions
- Identifiers Used to Denote Data Types

Scope

The functionality described in this guide can be used only in ICADVM18.1advanced nodes and advanced methodologies releases.

Licensing Requirements

For information on advanced node licensing in the Virtuoso design environment, see *License Requirements for Advanced Node Features* in *Virtuoso Software Licensing and Configuration User Guide*.

Related Documentation

What's New and KPNS

- *Virtuoso Multi-Patterning Technology What's New*
- *Virtuoso Multi-Patterning Technology Known Problems and Solutions*

Installation, Environment, and Infrastructure

- *Cadence Installation Guide*
- *Virtuoso Design Environment User Guide*
- *Virtuoso Design Environment SKILL Reference*
- *Cadence Application Infrastructure User Guide*

Technology Information

- *Virtuoso Technology Data User Guide*
- *Virtuoso Technology Data ASCII Files Reference*
- *Virtuoso Technology Data Constraints Reference*
- *Virtuoso Technology Data SKILL Reference*
- *Virtuoso Design Environment SKILL Reference*

Virtuoso Tools

- *Virtuoso Layout Viewer User Guide*
- *Virtuoso Layout Suite XL: Basic Editing User Guide*
- *Virtuoso Layout Suite XL: Connectivity Driven Editing Guide*

- *Virtuoso Layout Suite EXL Reference*
- *Virtuoso Concurrent Layout User Guide*
- *Virtuoso Design Planner User Guide*
- *Virtuoso Simulation Driven Interactive Routing User Guide*
- *Virtuoso Width Spacing Patterns User Guide*
- *Virtuoso RF Solution Guide*
- *Virtuoso Layout Suite SKILL Reference*

Additional Learning Resources

Video Library

The Video Library on the Cadence Online Support website provides a comprehensive list of videos on various Cadence products.

To view a list of videos related to a specific product, you can use the *Filter Results* feature available in the pane on the left. For example, click the *Virtuoso Layout Suite* product link to view a list of videos available for the product.

You can also save your product preferences in the Product Selection form, which opens when you click the *Edit* icon located next to *My Products*.

Virtuoso Videos Book

You can access certain videos directly from Cadence Help. To learn more about this feature and to access the list of available videos, see Virtuoso Videos.

Rapid Adoption Kits

Cadence provides a number of Rapid Adoption Kits that demonstrate how to use Virtuoso applications in your design flows. These kits contain design databases and instructions on how to run the design flow.

In addition, Cadence offers the following training courses that are relevant to Virtuoso Multi-Patterning Technology:

- Virtuoso Layout for Advanced Nodes
- Virtuoso Connectivity-Driven Layout Transition

Cadence also offers the following training courses on the SKILL programming language, which you can use to customize, extend, and automate your design environment:

- SKILL Language Programming Introduction
- SKILL Language Programming
- Advanced SKILL Language Programming

To explore the full range of training courses provided by Cadence in your region, visit Cadence Training or write to training_enroll@cadence.com.

Note: The links in this section open in a separate web browser window when clicked in Cadence Help.

Help and Support Facilities

Virtuoso offers several built-in features to let you access help and support directly from the software.

- The Virtuoso *Help* menu provides consistent help system access across Virtuoso tools and applications. The standard Virtuoso *Help* menu lets you access the most useful help and support resources from the Cadence support and corporate websites directly from the CIW or any Virtuoso application.
- The Virtuoso Welcome Page is a self-help launch pad offering access to a host of useful knowledge resources, including quick links to content available within the Virtuoso installation as well as to other popular online content.

  The Welcome Page is displayed by default when you open Cadence Help in standalone mode from a Virtuoso installation. You can also access it at any time by selecting *Help – Virtuoso Documentation Library* from any application window, or by clicking the *Home* button on the Cadence Help toolbar (provided you have not set a custom home page).

For more information, see Getting Help in *Virtuoso Design Environment User Guide.*

Customer Support

For assistance with Cadence products:

- Contact Cadence Customer Support

  Cadence is committed to keeping your design teams productive by providing answers to technical questions and to any queries about the latest software updates and training needs. For more information, visit https://www.cadence.com/support.

- Log on to Cadence Online Support

  Customers with a maintenance contract with Cadence can obtain the latest information about various tools at https://support.cadence.com.

Feedback about Documentation

You can contact Cadence Customer Support to open a service request if you:

- Find erroneous information in a product manual
- Cannot find in a product manual the information you are looking for
- Face an issue while accessing documentation by using Cadence Help

You can also submit feedback by using the following methods:

- In the Cadence Help window, click the *Feedback* button and follow instructions.
- On the Cadence Online Support Product Manuals page, select the required product and submit your feedback by using the *Provide Feedback* box.

Understanding Cadence SKILL

Cadence SKILL is a high-level, interactive programming language based on the popular artificial intelligence language, Lisp. It lets you customize and extend your design environment. Using SKILL you can validate the steps of your algorithm incrementally before incorporating them into a larger program.

For more information about the SKILL language, see Getting Started in the *SKILL Language User Guide*.

Using SKILL Code Examples

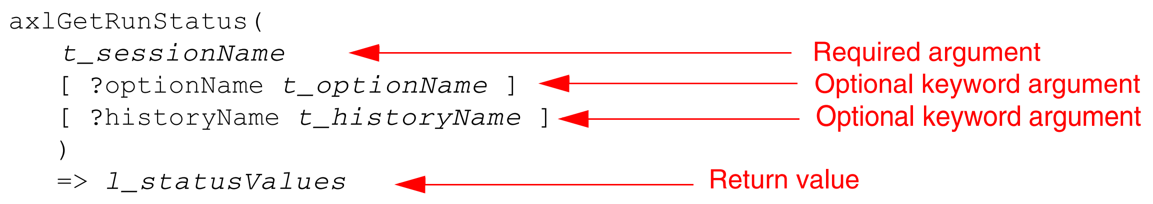
The SKILL APIs in this user manual are explained with illustrative code examples.

You can copy these examples from the manual and paste them directly into the Command Interpreter Window (CIW) or use the code in non-graphical SKILL mode.

Sample SKILL Code

The following code sample shows the syntax of a SKILL API that accepts three arguments.

axlGetRunStatus

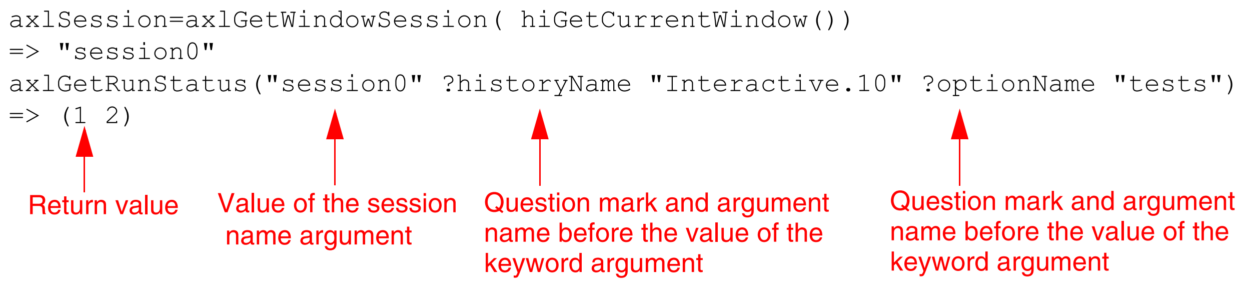


```
axlGetRunStatus(
    t_sessionName
    [ ?optionName t_optionName ]
    [ ?historyName t_historyName ]
    )
    => l_statusValues
```

The first argument `t_sessionName` is a required argument, where `t` signifies the data type of the argument. The second and third arguments `?optionName t_optionName` and `?historyName t_historyName` are optional keyword arguments (identified by a question mark), which are specified in name-value pairs and can be placed in any order during the function call.

The return value is the value that the SKILL API returns after evaluating the expression. In this case, it is a list of status values, *l_statusValues*.

Example



```
axlSession=axlGetWindowSession( hiGetCurrentWindow())
=> "session0"
axlGetRunStatus("session0" ?historyName "Interactive.10" ?optionName "tests")
=> (1 2)
```

Return value
Value of the session name argument
Question mark and argument name before the value of the keyword argument
Question mark and argument name before the value of the keyword argument

Accessing API Help

Quick reference information for SKILL APIs is available from the CIW and the SKILL API Finder. To access the reference information for a particular SKILL API, do one of the following:

- Type `help <function_name>` in the CIW.
- Type `startFinder ( [ ?funcName t_functionName ] )` in the CIW.
- Start the SKILL API Finder from the CIW by choosing *Tools – Finder* or type `cdsFinder` on the UNIX command line.

  In the *Search in* field of the displayed Cadence SKILL API Finder window, type the SKILL API name for which you want to display the help information and click *Go*.

  The matches for the searched SKILL API appear in the *Results* area.

  To view the complete documentation of the searched SKILL API, select the API name in the *Results* area and click the *More Info* button. The complete documentation of the selected SKILL API appears in a new Cadence Help window.

Typographic and Syntax Conventions

The following typographic and syntax conventions are used in this manual.

text	Indicates names of manuals, menu commands, buttons, and fields.
`text`	Indicates text that you must type as presented. Typically used to denote command, function, routine, or argument names that must be typed literally.
`z_argument`	Indicates text that you must replace with an appropriate argument value. The prefix (in this example, *`z_`*) indicates the data type the argument can accept and must not be typed.
|	Separates a choice of options.
`{ }`	Encloses a list of choices, separated by vertical bars, from which you **must** choose one.
`[ ]`	Encloses an optional argument or a list of choices separated by vertical bars, from which you **may** choose one.
`[ ?argName` *`t_arg`* `]`	Denotes a *key argument*. The question mark and argument name must be typed as they appear in the syntax and must be followed by the required value for that argument.
`...`	Indicates that you can repeat the previous argument. Used with brackets to indicate that you can specify zero or more arguments. Used without brackets to indicate that you must specify at least one argument.
`,...`	Indicates that multiple arguments must be separated by commas.
`=>`	Indicates the values returned by a Cadence® SKILL® language function.
`/`	Separates the values that can be returned by a Cadence SKILL language function.

If a command-line or SKILL expression is too long to fit within the paragraph margins of this document, the remainder of the expression is moved to the next line and indented. In code excerpts, a backslash (\ ) indicates that the current line continues on to the next line.

Identifiers Used to Denote Data Types

Data type identifiers are used to indicate the type of value required by an API argument. These data types are denoted by a single letter that is prefixed to the argument label and is separated from the argument by an underscore; for example, `t` is the data type in *`t_viewName`*. Data types and underscores are used only as identifiers; they must not be typed when specifying the argument in a function.

Prefix	Internal Name	Data Type
a	array	array
A	amsobject	AMS object
b	ddUserType	DDPI object
B	ddCatUserType	DDPI category object
C	opfcontext	OPF context
d	dbobject	Cadence database object (CDBA)
e	envobj	environment
f	flonum	floating-point number
F	opffile	OPF file ID
g	general	any data type
G	gdmSpecIlUserType	generic design management (GDM) spec object
h	hdbobject	hierarchical database configuration object
I	dbgenobject	CDB generator object
K	mapiobject	MAPI object
l	list	linked list
L	tc	Technology file time stamp
m	nmpIlUserType	nmpIl user type
M	cdsEvalObject	cdsEvalObject
n	number	integer or floating-point number
o	userType	user-defined type (other)
p	port	I/O port
q	gdmspecListIlUserType	gdm spec list

Prefix	Internal Name	Data Type
r	defstruct	defstruct
R	rodObj	relative object design (ROD) object
s	symbol	symbol
S	stringSymbol	symbol or character string
t	string	character string (text)
T	txobject	transient object
u	function	function object, either the name of a function (symbol) or a lambda function body (list)
U	funobj	function object
v	hdbpath	hdbpath
w	wtype	window type
sw	swtype	subtype session window
dw	dwtype	subtype dockable window
x	integer	integer number
y	binary	binary function
&	pointer	pointer type

For more information, see *Cadence SKILL Language User Guide*.

1 Introduction

This chapter provides a general overview of multi-patterning lithography, its challenges, and how Virtuoso® Multi-Patterning Technology (MPT) can help you meet the needs for advanced node designs.

- Multi-Patterning Lithography on page 20
- Introducing Virtuoso Multi-Patterning Technology on page 22

Multi-Patterning Lithography

For a number of years, IC manufacturing has been pushing the limits of optical lithography to make silicon with features smaller than the conventional ones by employing various resolution enhancement techniques (RET). As the manufacturing equipment and lithography process struggle to keep up with diminishing feature dimensions, their resolution capabilities have fallen further and further behind the target minimum feature size per each advanced node. As a result, optical lithography has finally become unable to print shapes on silicon with a single mask in a single pass starting at 20nm. Available RET/optical proximity correction (OPC) techniques are not able to yield expected feature sizes in close proximity reliably.

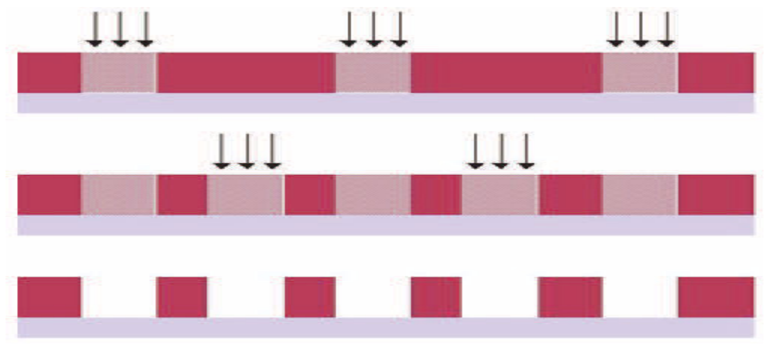
The solution to this problem is to use a technique that has existed for years in the photographic industry known as *multi-patterning* in which two or more mask processes are used to manufacture each design layer. For each design layer, the layout geometry must be decomposed onto separate masks (or colors), typically based on proximity to the nearest shape. To represent a particular mask at the design level, a "color" is associated with each layout shape to indicate the mask used to print the shape. Color decomposition is the process of determining the mask used to print each of the shapes on the same drawn layer while ensuring that none of the shapes assigned to the same mask are too close together to be printed correctly. After a successful decomposition, all geometries with the same color must be at least same mask spacing apart, where the same mask spacing value is defined by the foundry rules. Shapes with different colors can be closer together than same mask spacing because they are printed by different masks, and therefore, do not interact optically like the shapes on the same mask.

Multi-Patterning Technology

For advanced nodes, two or more lithography masks are needed and Multi-Patterning Technology (MPT) is used. There are two popular techniques for applying multiple patterning:

- Litho-Etch-Litho-Etch (LELE)

Each mask exposure is followed by an etching step. A simple double-pattern diagram is shown here.



- Self-Aligned Double Patterning (SADP)

  The first mask uses side walls (edges) of a feature to define the desired patterns, followed by a second "trim" mask to block out unwanted patterns.

Introducing Virtuoso Multi-Patterning Technology

Virtuoso® Multi-Patterning Technology (MPT) is used for multi-patterning lithography, such as Litho-Etch-Litho-Etch (LELE) and Self-Aligned Double Patterning (SADP), and uses different colors to represent up to four masks for each drawn layer. For LELE, the colors directly map to the masks. For SADP, this method is effective in determining whether a design can be successfully decomposed during the mask-making step.

Virtuoso MPT supports many coloring schemes, including the following:

- Schematic-Driven Layout Support of Mask Name

  In the schematic view, critical nets are identified and placed in a *net class*. A *Same Mask* pre-coloring constraint can be assigned to the net class to specify that all the shapes for the nets in the net class must be on the same mask for each routing layer. In addition, the mask name can be specified. During stream out, shapes for pre-colored nets are locked to their current color. For more information, see Net-Based Pre-Coloring Flow.

- Interactive Coloring in Layout

  In the layout view, there are many ways to color shapes and view color properties, such as using the Properties Editor, Palette Assistant, Dynamic Selection, and the Multiple Patterning toolbar.

- Coloring using width spacing patterns for track-based routing

  For more information, see *Virtuoso Width Spacing Patterns User Guide*.

- Coloring using an external engine

  For more information, see Fully Colored Backannotation Flow.

These schemes can be used in combination with color designs.

Virtuoso MPT also supports methods for the following:

- Checking for multi-patterning violations in layout

  Pegasus Interactive and Virtuoso DRD can be used to identify same-mask spacing violations. For more information, refer to Checking for Multiple Patterning Violations.

- Fixing multi-patterning violations

  Multi-patterning violations may be fixed by moving, splitting, or stretching shapes, or by polygon splicing (stitching). For more information, refer to Fixing Multiple Patterning Violations.

- Abstract generation

  Virtuoso® Abstract Generator is a library modeling tool that has been enhanced to generate a color-annotated abstract cellview from a color-annotated layout cellview, as described in Multi-Patterning Technology Support in Abstract Generator in *Virtuoso Abstract Generator User Guide.*

- XStreamIn/Out

  Color mapping functionality has been added to XStream In and XStream Out translators, as described in Using XStream.

2

Getting Started

The following topics are included in this chapter:

- Prerequisites
    - Specifying the Number of Masks Per Layer
    - Specifying Same-Mask and Diff-Mask Spacing Constraints
    - Specifying Coloring Purposes
    - Example of ASCII Technology File Data for MPT
- Customizing Your Environment

Prerequisites

Specifying the Number of Masks Per Layer

By default, there is one mask per layer. For layers that support more than one mask, you **must** specify the number of masks in a column of the DFII ASCII technology file `techLayers` section.

When there is more than one mask for a layer, the masks are represented by colors in Virtuoso:

- Color1 (`mask1Color`)
- Color2 (`mask2Color`)
- Color3 (`mask3Color`)
- Color4 (`mask4Color`)
- Color5 (`mask5Color`)
- Color6 (`mask6Color`)
- Color7 (`mask7Color`)
- Color8 (`mask8Color`)
- No color (`gray` or `grayColor`)

The color choices for a layer are dependent on the number of masks that have been set for the layer in the technology file.

For an example of how to set the number of masks for a layer in an ASCII technology file, refer to Example of ASCII Technology File Data for MPT.

Specifying Same-Mask and Diff-Mask Spacing Constraints

For active color management, same-mask and different mask (diff-mask) spacing constraints must be set in the technology database. Table 2-1 lists the supported constraints and parameters for mask coloring.

Table 2-1 Same-Mask and Diff-Mask Spacing Constraints

Spacing Check	Constraint	Constraint Parameters
Side-to-Side	minSpacing	'sameMask
Corner-to-Corner	minSpacing (PRL<0)	'sameMask
	minCornerSpacing	'sameMask
End-to-Edge	minEndOfLineSpacing	'sameMask 'diffMask
	minEndOfLineExtensionSpacing	'sameMask
Edge-to-edge	minSpanLengthSpacing	'sameMask
Joint corner-to-joint corner	minJointCornerSpacing	'sameMask
End-to-corner or edge	endOfLineKeepout	'mask1 | 'mask2 | 'mask3
Via-to-Via	minViaSpacing	'sameMask
Cut class-to-cut class	minCutClassSpacing	'sameMask

For detailed information on these constraints, refer to *Virtuoso Technology Data Constraints Reference*.

For an example of same-mask spacing constraints in an ASCII technology file, refer to Example of ASCII Technology File Data for MPT.

Specifying Coloring Purposes

By default, coloring will be applied only to shapes on the `drawing` and `pin` purposes, all user-defined purposes, and purposes for which one of these is the parent purpose. A shape on any other predefined purpose will not be colored unless its purpose is specified by the `coloredPurposeTypes` environment variable. Alternatively, if the `explicitColoredPurposes` environment variable is a non-empty string, coloring will be applied only to shapes on the purposes specified by that environment variable; in this case, the `coloredPurposeTypes` environment variable will be ignored. For the list of predefined purposes, see Predefined Purposes in *Virtuoso Technology Data ASCII Files Reference*.

Note: If the MPT constraint group is defined, the purposes specified in the `coloredPurposeTypes` and `explicitColoredPurposes` environment variables are not considered for coloring. Even if no valid purposes are defined in the constraint group definition, the default purposes are considered for coloring and environment variables are not considered.

Example of ASCII Technology File Data for MPT

In this example,

- The number of masks per layer is set for each layer that has more than one mask.
- The member constraint groups, `minEndOfLineSpacingAndSameMaskCG`, `minSpacingTableAndSameMaskCG`, and `minViaSpacingAndSameMaskCG`, are referenced in the **foundry** constraint group.

First, the number of masks per layer is set.

```
techLayers(
;( LayerName  Layer#  Abbreviation )
;( ---------  ------  ------------ )
;( NEW to support MPT              [# of masks] )
;(                                  ----------  )
 ( V0         25      V0              2 )
 ( Metal1     30      M1              2 )
 ( Via1       32      V1              2 )
 ( Metal2     34      M2              3 )
```

Next, the member constraint groups are created.

The `minEndOfLineSpacingAndSameMaskCG` member group sets the `minEndOfLineSpacing` constraint for end-to-end and end-to-side same-mask spacing ANDed with the foundry `minEndOfLineSpacing` rule.

```
;;; Sub-constraint group for same-mask end-of-line spacing AND group
;( group                    )
;( -----    ------------ )
( "minEndOfLineSpacingAndSameMaskCG" nil nil 'and
   spacings(

    ; This is same-mask end-to-end and end-to-side spacing
    ; end-to-end same-mask spacing is the endToEndSpace parameter value.
    ; end-to-side same-mask spacing is the minEndOfLineSpacing constraint value.
    ( minEndOfLineSpacing "Metal1"
       'sameMask
       'paraEdgeCount 0
       'width 0.08
       'distance 0.025
       'endToEndSpace 0.13
       'otherEndWidth 0.08
      0.13 'description "Same-Mask end-to-side and end-to-end spacing" )

    ; This is the foundry rule, sameMask is not set
    ( minEndOfLineSpacing "Metal1"
        'width 0.09
        'distance 0.025
        'paraEdgeSpace 0.025
        'paraEdgeWithin 0.09
        'paraEdgeCount 2
      0.09 'description "Minimum Metal1 End of Line Spacing")
   ) ;spacings
) ;minEndOfLineSpacingAndSameMaskCG
```

AND group

End-to-End spacing

End-to-Side spacing

The `minSpacingTableAndSameMaskCG` member group sets the `minSpacing` constraint for side-to-side and corner-to-corner same-mask spacing ANDed with the foundry `minSpacing` rules.

```
;;; Sub-constraint group for same-mask spacing table AND group
;( group                    )
;( -----    ------------ )
( "minSpacingTableAndSameMaskCG"  nil nil 'and
   spacingTables(

    ; This is same-mask corner-to-corner and side-to-side spacing
    ( minSpacing "Metal1"
    ;     width             length             spacing
      (("width" nil nil "length" nil nil )  'sameMask )

       ((0.06              -0.035   )          0.1
        (0.06              -0.03    )          0.1
        (0.06               0.6     )          0.12
        (0.08              -0.035   )          0.1
        (0.08              -0.03    )          0.12
        (0.08               0.6     )          0.14
        (0.12              -0.035   )          0.1
        (0.12              -0.03    )          0.14
        (0.12               0.6     )          0.16
      )
    )
    ; This is the foundry rule, sameMask is not set
    ( minSpacing "Metal1"
    ;     width             length             spacing
      (("width" nil nil "length" nil nil )       )
       ((0.0005             0.0005        )    0.06
        (0.0005             0.3205        )    0.06
         ...
       )
     )
    ); spacingTables
); minSpacingTableAndSameMaskCG
```

AND group

Corner-to-Corner spacing

All others in this table are width/PRL-based **Side-to-Side** spacing.

sameMask not set. This is the foundry minSpacing rule.

The `minViaSpacingAndSameMaskCG` member group sets the `minViaSpacing` constraint for via-to-via same-mask spacing ANDed with the foundry `minViaSpacing` rules.

```
;;; Sub-constraint group for same-mask spacing table AND group
;( group             )
;( -----    ------------ )
( "minViaSpacingAndSameMaskCG"  nil nil 'and
   spacings(
    ( minViaSpacing "Via1"
         'centerToCenter
         'sameMask
         1.5 )
    ( minViaSpacing "Via1"
         1.0 )
   ) ;spacings
) ;minViaSpacingAndSameMaskCG
```

AND group

Same Mask **Via-to-Via** spacing, measured center-to-center

sameMask not set.
This is the foundry minViaSpacing rule.

Finally, the member groups are specified in the foundry constraint group.

;;; Specification of member constraint groups in the foundry constraint group

```
;( group                    [override] )
;( -----    ------------ )
( "foundry"          nil
   memberConstraintGroups(
     ; listed in order of precedence
     "minEndOfLineSpacingAndSameMaskCG"
     "minSpacingTableAndSameMaskCG"
     "minViaSpacingAndSameMaskCG"
     ); memberConstraintGroups
```

Customizing Your Environment

There are a number of factors that determine how data will be displayed and handled, and which Virtuoso Multi-Patterning Technology GUI elements will appear. These factors can be set using environment variables, SKILL functions, and the Multiple Patterning Options form.

Shapes can be colored two ways:

- Manually using the Virtuoso tools described in Interactive Coloring in Layout
- Automatically using the multiple patterning color engine

  The color engine can automatically color connected shapes on the same layer and change the color of shapes to avoid same-mask spacing violations.

This section includes the following topics:

- Enabling the Multiple Patterning Color Engine
  - Setting the Coloring Method
- Using the Multiple Patterning Options Form
- Customizing Displayed Coloring

Enabling the Multiple Patterning Color Engine

By default, the color engine is not enabled.

To use the color engine:

1. Do one of the following:
   - Type the following in the CIW or in your `.cdsinit` file:

     ```
     mptActivate( t )
     ```
   - Use the Multiple Patterning toolbar, as described in Turning the Multiple Patterning Color Engine On and Off.
2. Set the coloring method, as described in Setting the Coloring Method.

Setting the Coloring Method

The coloring method applies when the multiple patterning color engine is enabled and can be set globally for a session, and by layer for a technology database or a cellview. The valid coloring methods are: *interactive* and *managed*. For both of these methods, the color engine automatically colors connected shapes on the same layer. When the coloring method is *managed*, the color engine can also change the color of shapes to avoid same-mask spacing violations.

Global Coloring Method

To set the global coloring method for the session, do one of the following:

➡ Type the following in the CIW input line or in your `.cdsinit` file:

  `mptSetDefaultColoringMethod( t_coloringMethod )`

  where *`t_coloringMethod`* is `"interactive"` or `"managed"`.

  For example,

```
mptSetDefaultColoringMethod( "managed" )
```

➡ Specify the *Default Coloring Method*, as described in Using the Multiple Patterning Options Form.

Layer Coloring Method

To set the coloring method for a layer,

➡ Type the following in the CIW input line or in your `.cdsinit` file.

  `mptSetLayerColoringMethod( d_objID l_layers t_coloringMethod )`

  where *`d_objID`* is the object identifier for the cellview or the technology database, *`l_layers`* is a list of layers, and *`t_coloringMethod`* is `"interactive"` or `"managed"`.

Coloring Method Precedence

- By default, the global coloring method is used for all layers.
- The layer coloring method for the technology database takes precedence over the global coloring method.
- The layer coloring method for a cellview takes precedence over the layer coloring method for the technology database.

Getting the Coloring Method

To get the global default coloring method, see the *Default Coloring Method*, as described in Using the Multiple Patterning Options Form, or use the `mptGetDefaultColoringMethod` SKILL function.

```
t_coloringMethod = mptGetDefaultColoringMethod()
```

where *`t_coloringMethod`* is `"interactive"` or `"managed"`.

To get the coloring method for a layer, use the `mptGetLayerColoringMethod` SKILL function.

```
t_coloringMethod = mptGetLayerColoringMethod( d_cellviewID t_layerName )
```

where *`d_cellviewID`* is the object identifier for the cellview, *`t_layerName`* is the layer name and *`t_coloringMethod`* is `"interactive"` or `"managed"`.

To get the coloring method for the session, the design, and the layers for which the coloring method is set, use the `mptGetLayerColoringMethod` SKILL function.

In the following example, the output is shown in blue.

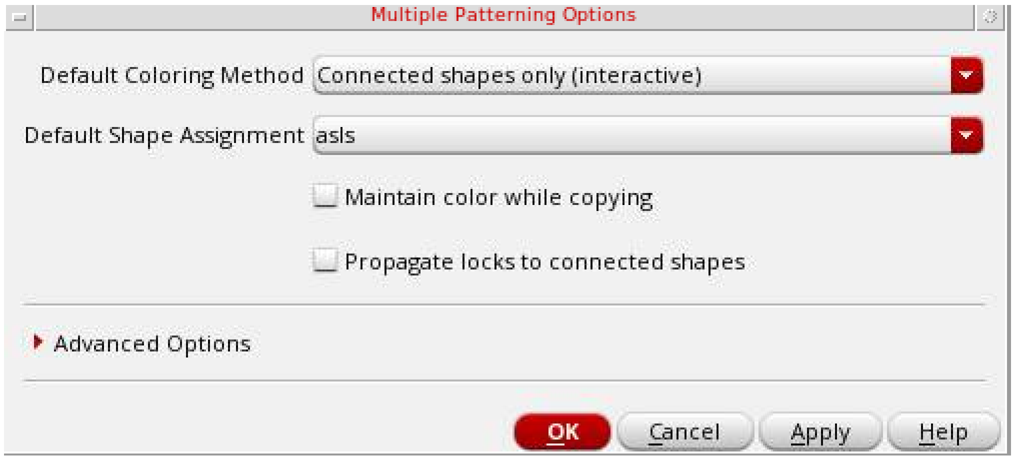
```
mptGetLayerColoringMethod( cv )
Session coloring method : interactive
Design coloring : interactive
Layer Metal2: managed
t
```

Using the Multiple Patterning Options Form

To change the default multiple patterning environment variable settings:

1. Choose *Options – Multiple Patterning*.

The Multiple Patterning Options form appears.



Default Coloring Method: Connected shapes only (interactive)
Default Shape Assignment: asls
☐ Maintain color while copying
☐ Propagate locks to connected shapes

Tip

Hover the pointer over a field to see the tooltip for the field.

2. Choose the *Default Coloring Method.*

- *Connected shapes only (interactive)*
- *Connected shapes & color spacing rules (managed)*

The coloring method applies only when the color engine is enabled. For both interactive and managed coloring methods, the color engine automatically colors connected shapes on the same layer. When the coloring method is *managed*, the color engine can also change the color of shapes to avoid same-mask spacing violations. For information on how this setting affects coloring, see Using the Multiple Patterning Color Engine.

3. Choose the *Default Shape Assignment.*

- *gray*: Shapes are not colored.
- *layerDefault*: Shapes are assigned to the default color for the layer.
- *random*: Shapes are randomly assigned to a valid mask color for the layer.
- *asls*:
  - Shapes with a color assignment are unchanged unless a color violation exists. If a color violation exists, the color engine can change the color of a shape.
  - Shapes without a color assignment but that are same-mask spacing or less from another same-layer shape are colored using the layer default color, if set, or randomly colored. Otherwise, gray shapes remain gray.

4. Choose whether to *Maintain color while copying.*

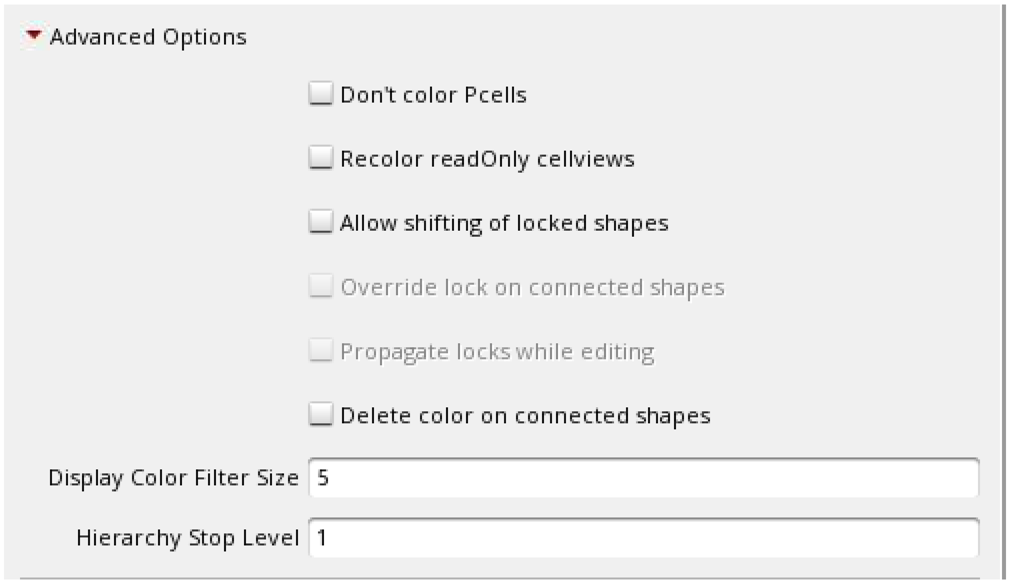
   When enabled, the coloring information from the source objects is copied *as-is* to the corresponding destination objects when copying or using Make Cell.

5. Choose whether to *Propagate locks to connected shapes.*

   When enabled, locks are automatically propagated to connected shapes when the lock is initiated from the Multiple Patterning toolbar. For more information on this option, see Color Locking on Connected Shapes.

6. To specify *Advanced Options*, click the expand (triangle) button.

   The *Advanced Options* appear.



Advanced Options

- [ ] Don't color Pcells
- [ ] Recolor readOnly cellviews
- [ ] Allow shifting of locked shapes
- [ ] Override lock on connected shapes
- [ ] Propagate locks while editing
- [ ] Delete color on connected shapes

Display Color Filter Size: 5

Hierarchy Stop Level: 1

   a. Choose *Don't color Pcells* to prevent the recoloring of Pcells when running *ReColor All* or *Update Color* from the Multiple Patterning toolbar or using the `mptReColor` or `mptUpdateColor` SKILL function.

   b. Choose *Recolor readOnly cellviews* to recolor read-only cellviews when running *ReColor All* or *Update Color* from the Multiple Patterning toolbar or using the `mptReColor` or `mptUpdateColor` SKILL function. If not selected, only editable cellviews are recolored by those Multiple Patterning toolbar and SKILL functions.

   c. Choose *Allow shifting of locked shapes* to allow shifting of color-locked shapes. By default, shapes must be unlocked before shifting colors.

   d. Choose *Override lock on connected shapes* to allow color-locked shapes to change color to avoid color conflicts with connected shapes when a lock is initiated from the Multiple Patterning toolbar. By default, color-locked shapes cannot change color to avoid color conflicts. For more information on this option, see Color Locking on Connected Shapes.

e. Choose *Propagate locks while editing* to automatically propagate locks to connected shapes on the same layer while editing. When this option is disabled and shapes are connected while editing (for example, when a shape is moved to connect to another shape), only the color of the shape can be propagated. When this option is enabled, the color state (lock) can also be propagated. For more information on this option, see Color Locking on Connected Shapes.

f. Choose *Delete color on connected shapes* to delete the color on the shapes connected to the selected shape

g. Specify *Display Color Filter Size* for the minimum shape size, in pixels, for which the color of the shape will be displayed. The color of shapes smaller than this size will not be displayed.

h. Specify *Hierarchy Stop Level* for the level to which shapes will be considered when coloring. A value of `1` considers only top-level and level-1 shapes. A value of `0` considers only shapes on the current level.

7. Click *OK* or *Apply*.

The environment variables that are supported by the Multiple Patterning Options form are listed in the following table.

Table 2-2 Multiple Patterning Options Mapping to Environment Variables

Field	Environment Variable	Default
Default Coloring Method	`defaultColoringMethod`	`connectedShapes`
Default Shape Assignment	`unclusteredShapeColor`	`asIs`
Maintain color while copying	`copyMPAttributes`	`nil` (OFF)
Propagate locks to connected shapes	`propagateLocksToConnectedShapes`	`nil`
Don’t color Pcells	`dontColorPCells`	`nil`
Recolor readOnly cellviews	`reColorReadOnlyCellView`	`t`
Allow shifting of locked shapes	`allowLockShiftOverride`	`nil`

Table 2-2 Multiple Patterning Options Mapping to Environment Variables

Field	Environment Variable	Default
Override lock on connected shapes	`overrideLockOnConnectedShapes`	`nil`
Propagate locks while editing	`autoPropagateLock`	`nil`
Display Color Filter Size	`coloringFilterSize`	`5.0`
Hierarchy Stop Level	`extractorStopLevel`	`1`

For more information on other MPT environment variables, see List of Virtuoso MPT Environment Variables.

Customizing Displayed Coloring

The default coloring is shown in Color Representations. You can customize the coloring for Multiple Patterning data by defining color display packets that are sub-packets for the layer-purpose pair (LPP).

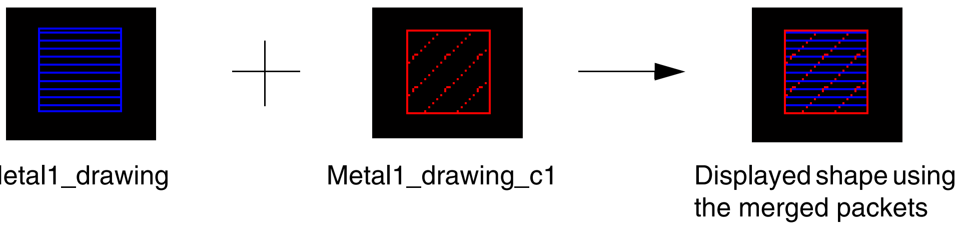
1. In the `techDisplays` section of the ASCII technology file, specify the packet name of the layer-purpose pair of interest.
2. Use the following names for the sub-packets in the **display.drf** file, where the coloring information is specified.

<packet_name>	Display packet for the uncolored LPP
<packet_name>`_c1`	Display packet for `mask1Color` unlocked LPP
<packet_name>`_c2`	Display packet for `mask2Color` unlocked LPP
<packet_name>`_c1L`	Display packet for `mask1Color` locked LPP
<packet_name>`_c2L`	Display packet for `mask2Color` locked LPP
<packet_name>`_black`	Display packet for a `blackColor` blockage object
<packet_name>`_multi`	Display packet for a `multiColor` blockage object

Note: Not all sub-packets are required. Since only blockage objects can be assigned `blackColor` or `multiColor`, only blockage display packets should define `_black` and `_multi` sub-packets. Blockages also cannot be color locked, so `_c1L` and `_c2L` sub-packets for a blockage display style are not needed.

Displaying Colored Shapes

When custom coloring is specified, colored shapes are displayed using the merged colored and uncolored display packets. In the following example, the uncolored packet and the mask1Color unlocked packet are shown graphically with the resultant colored shape.



To display colored shapes using only the colored display packets, set the mergeColoredPacket environment variable to `nil`.

Example

In the ASCII technology file,

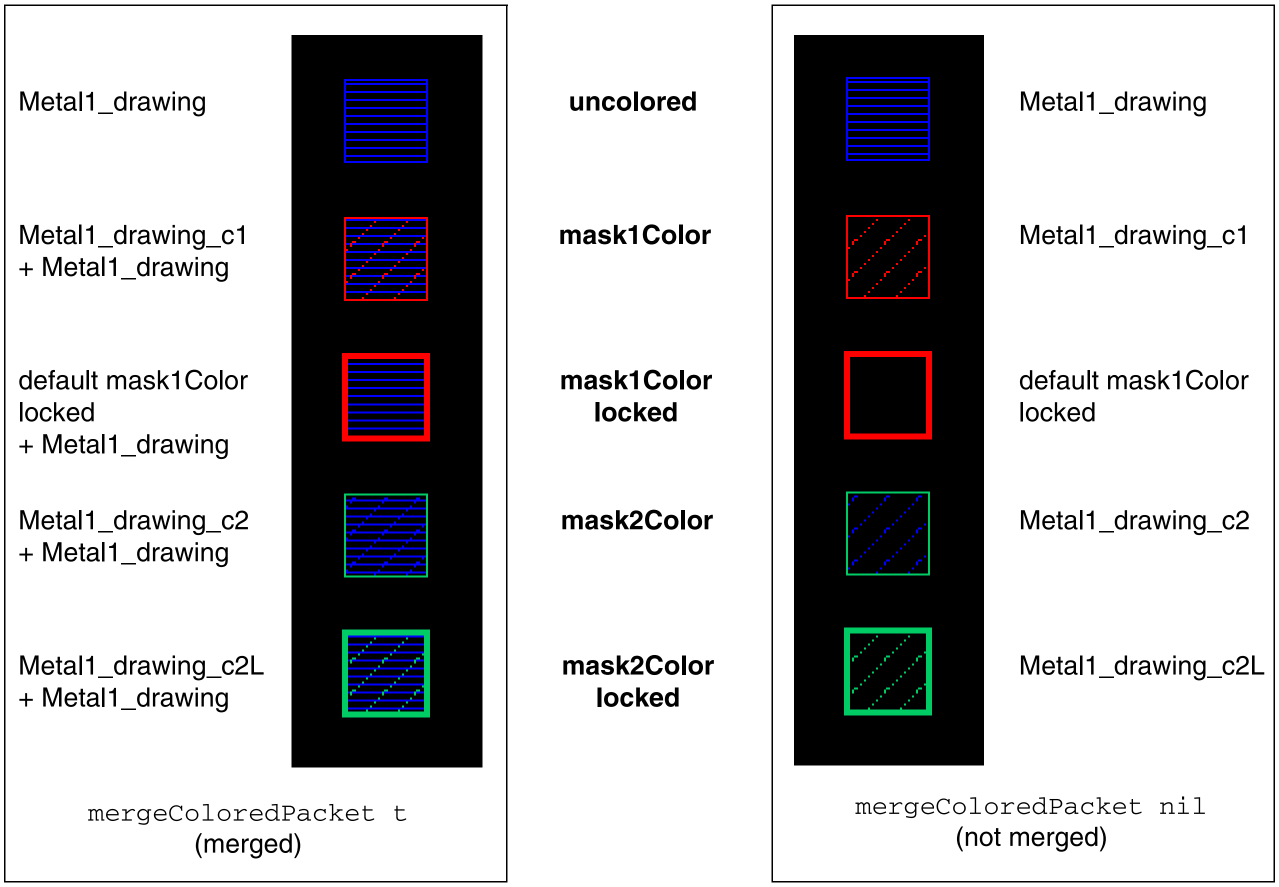
```
techDisplays(
;( LayerName Purpose  Packet …)
 ( Metal1    drawing  M1_drawing  …)
 ( Metal1    blockage M1_blockage …)
 ( Via1      drawing  V1_drawing  …)
  …
 ) ;techDisplays
```

In the `display.drf`,

```
drDefinePacket(
;(DispName PacketName      Stipple    LineStyle Fill   Outline [FillStyle])
 (display  M1_drawing        hLine    solid     blue   blue    outlineStipple)
 (display  M1_drawing_c1     stipple0 solid     red    red     outlineStipple)
 (display  M1_drawing_c2     stipple0 solid     blue   green   outlineStipple)
 (display  M1_drawing_c2L    stipple0 thickLine green  green   outlineStipple)
 (display  M1_blockage       hLine    solid     blue   blue    outlineStipple)
 (display  M1_blockage_c1    hLine    solid     blue   red     outlineStipple)
 (display  M1_blockage_c2    hLine    solid     blue   iceblue outlineStipple)
 (display  M1_blockage_black hLine    solid     blue   lilac   outlineStipple)
 (display  M1_blockage_multi hLine    solid     blue   purple  outlineStipple)
 (display  V1_drawing        solid    thickLine orange orange  X)
 (display  V1_drawing_cl     solid    thickLine orange red     outlineStipple)
 (display  V1_drawing_c2     solid    thickLine orange green   outlineStipple)
 …
 )
```

There is no display packet specified for `M1_drawing_c1L`. Therefore, the default mask1Color locked display packet is used which has a thick red outline with no stipple.

The graphical representations for shapes using this `display.drf` for the two mergeColoredPacket environment variable settings are shown below.


Metal1_drawing
uncolored
Metal1_drawing
Metal1_drawing_c1 + Metal1_drawing
mask1Color
Metal1_drawing_c1
default mask1Color locked + Metal1_drawing
mask1Color locked
default mask1Color locked
Metal1_drawing_c2 + Metal1_drawing
mask2Color
Metal1_drawing_c2
Metal1_drawing_c2L + Metal1_drawing
mask2Color locked
Metal1_drawing_c2L
mergeColoredPacket t (merged)
mergeColoredPacket nil (not merged)

3

Net-Based Pre-Coloring Flow

This chapter describes the pre-coloring flow for identifying the critical nets in the schematic to be routed on one mask for each layer.

The following sections are included:

- Overview
- Requirements
- Identifying the Critical Nets
- Assigning the Pre-Coloring Constraints
- Viewing Pre-Colored Nets in Layout
- Streaming Out Pre-Colored Nets

Overview

- In the schematic view, critical nets are identified and placed in a *net class*.
- Pre-coloring and spacing constraints are assigned to a reflexive constraint group (Within Group) for the net class.
- The pre-coloring and spacing constraints from the schematic for the net class must be propagated to the layout to ensure that critical nets use the constraint settings for coloring in layout.
- During stream out, shapes for the pre-colored nets are locked to their current color, based on the pre-coloring and spacing constraints.

The pre-coloring flow steps are described in the following sections:

- Identifying the Critical Nets
- Assigning the Pre-Coloring Constraints
- Viewing Pre-Colored Nets in Layout
- Streaming Out Pre-Colored Nets

Video

For a video overview of this feature, see Using the Net-Based Pre-Coloring Flow on Cadence Online Support.

Requirements

There must at least two masks set for a routing layer in the technology file. For more information, refer to Specifying the Number of Masks Per Layer.

Identifying the Critical Nets

In the schematic view,

1. Select one or more nets.
2. Open the Constraint Manager assistant by choosing *Window – Assistants – Constraint Manager.*
3. In the *Constraint Creation* pull-down menu of the Constraint Manager toolbar, choose *Routing – Net Class*.

A new net class containing the selected nets appears in the constraint view.

Assigning the Pre-Coloring Constraints

In the Constraint Manager constraint view, set the following in the *Within Group* for the net class that contains the critical nets:

- Spacing constraints and/or pre-coloring spacing constraint groups
- *Same Mask* pre-coloring constraint `true` specifies that all the shapes for the nets in the net class must be on the same mask for each routing layer.
- (optional) *Mask Name* ensures that shapes will be on the specified color mask for each routing layer for the nets in the net class.

For details, see Specifying the Color Mask for Critical Nets in Schematic in *Virtuoso Unified Custom Constraint User Guide*.

Viewing Pre-Colored Nets in Layout

Updating Layout Constraints

To update constraints in the layout to match those set in the schematic, do one of the following in the layout view:

- Choose *Connectivity – Update – Layout Constraints* on the menu bar.
- Click *Update Layout Constraints* on the Constraint Manager toolbar.

Important

The layout constraints must be updated whenever the schematic *Same Mask* or *Mask Name* setting is changed in the Constraint Manager GUI and when the related constraints are changed using SKILL.

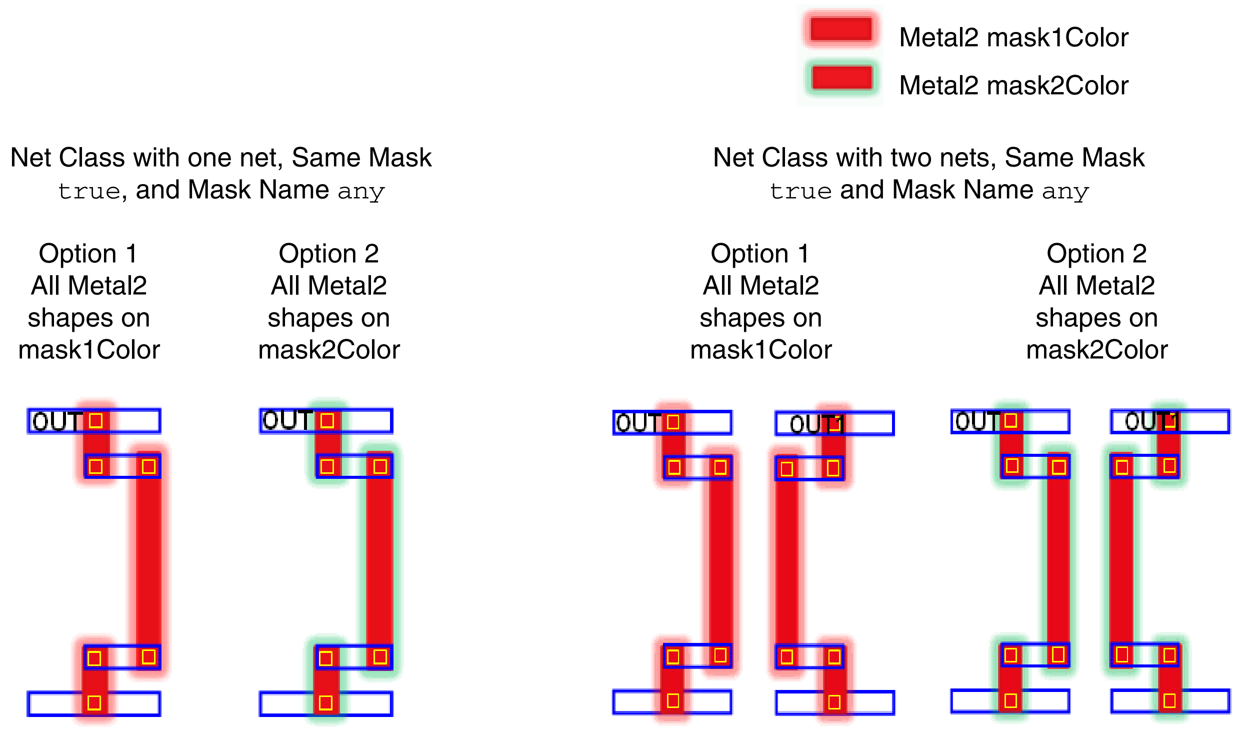
Tools, such as the Virtuoso color engine and router, will honor the *Same Mask* pre-coloring and spacing constraints for the special net class.

Same Mask Pre-Coloring

When the *Same Mask* pre-coloring constraint is `true` and the *Mask Name* pre-coloring constraint is `any` for a net class, all the shapes on the same routing layer for the nets in the net class are assigned to the same color mask. If the `propagateAnySameMaskState` environment variable is set to `t`, then all the shapes on the same routing layer for the nets in the net class are locked when one shape is locked. By default, locks are not propagated. Figure 3-1 shows examples of net classes consisting of one and two nets with *Same Mask* `true`. All shapes on Metal2 appear on the same color mask for all the nets of the net class.

When the *Same Mask* pre-coloring constraint is `true` and the *Mask Name* pre-coloring constraint is other than `any`, all the shapes on the same routing layer for the nets in the special net class are assigned to the specified *Mask Name* color and the color state is *locked*.

Figure 3-1 Same Mask Pre-Coloring Examples


Metal2 mask1Color
Metal2 mask2Color
Net Class with one net, Same Mask true, and Mask Name any
Net Class with two nets, Same Mask true and Mask Name any
Option 1 All Metal2 shapes on mask1Color
Option 2 All Metal2 shapes on mask2Color
Option 1 All Metal2 shapes on mask1Color
Option 2 All Metal2 shapes on mask2Color
OUT
OUT
OUT
OUT1
OUT
OUT1

Streaming Out Pre-Colored Nets

During stream out, shapes for the pre-colored nets are locked to their current color, based on the spacing constraints and the pre-coloring method.

To stream out pre-colored nets, follow the procedure in Exporting Stream Files with Coloring.

4

Coloring in Layout

There are several ways to color shapes in the layout. Shapes can be colored manually, as described in Interactive Coloring in Layout, or automatically Using the Multiple Patterning Color Engine. In track-based routing, shapes are colored based on their position relative to colored tracks, as described in Track-Based Coloring.

The following topics are included in this chapter:

- Interactive Coloring in Layout
- Using the Multiple Patterning Toolbar
- Using the Multiple Patterning Color Engine
- Track-Based Coloring
- Migrating from the Multiple Patterning Assistant
- Saving and Restoring Data
- Limitations

Interactive Coloring in Layout

Several Virtuoso tools can be used to color shapes and view coloring in Layout:

- Palette Assistant
    - Creating Colored Shapes
    - Controlling the Visibility and Selectability of Colored Data

  For more information, see MPT Support in *Virtuoso Layout Suite L User Guide*.
- Property Editor
    - Inspecting Colored Data
    - Coloring Existing Shapes
    - Color Locking

  For more information, see Interactively Setting Color on a Selected Object in *Virtuoso Layout Suite L User Guide*.
- Dynamic Selection Assistant
    - Inspecting Colored Data

  For more information, see Dynamic Selection in *Virtuoso Layout Suite L User Guide*.
- Multiple Patterning Toolbar
    - Showing and Hiding Color
    - Coloring Existing Shapes
    - Color Shifting
    - Recoloring Selected Colors
    - Color Locking
    - Removing Color from Shapes

  For more information, refer to Using the Multiple Patterning Toolbar.
- Show Selection Info Toolbar
    - Inspecting Colored Data

  For more information, refer to Displaying Preselect and Selected Object Information in *Virtuoso Layout Suite L User Guide*.

- Virtuoso Space-based Router

  For information on creating colored wires and buses, see Interactive Colored Routing in *Virtuoso Interactive and Assisted Routing User Guide*.

  (message URL ../weuser/interactiveColoredRouting.html#firstpage)

Video

For an overview of this feature, see Basic Interactive (Manual) Coloring.

The advanced coloring features are available only when the multiple patterning color engine is enabled and are dependent on the coloring method (interactive or managed).

Table 4-1 Advanced Coloring Features (Color Engine ON)

Feature	Interactive	Managed
Hierarchical Coloring of Connected Shapes	Automatic	Automatic
Automatic Color Shifting	no	Yes, based on same-mask constraint settings

For more information, refer to Using the Multiple Patterning Color Engine.

Color Representations

Table 4-2 shows the default color representations for the first four masks. Shapes without color assignments are considered to be “gray” and are shown with no outline by default.

Table 4-2 Default Colors

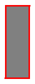
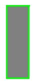
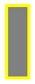
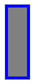
Color1 mask1Color (red outline)	Color2 mask2Color (green outline)	Color3 mask3Color (yellow outline)	Color4 mask4Color (blue outline)	Description	Outline Style
				Unlocked	Thin line

Table 4-2 Default Colors

🟥	🟩	🟨	🟦	Locked	Thick line

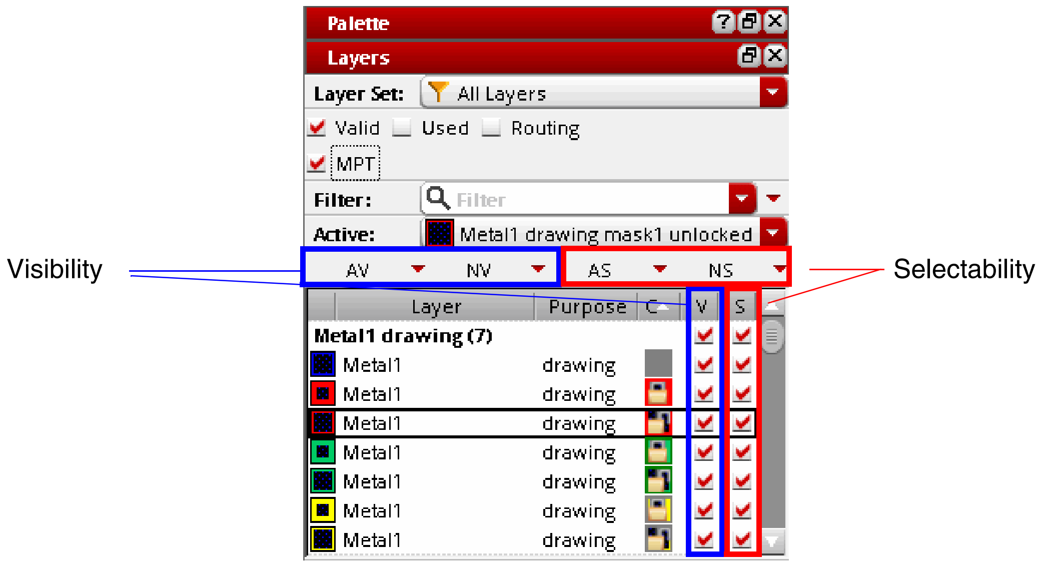
For information on how to change mask colors and outline styles, refer to Customizing Displayed Coloring.

Creating Colored Shapes

To add colored shapes to a design, follow the instructions in Drawing Shapes with Color in *Virtuoso Layout Suite L User Guide*.

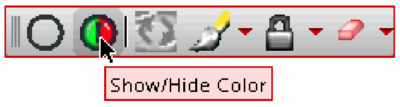
Controlling the Visibility and Selectability of Colored Data

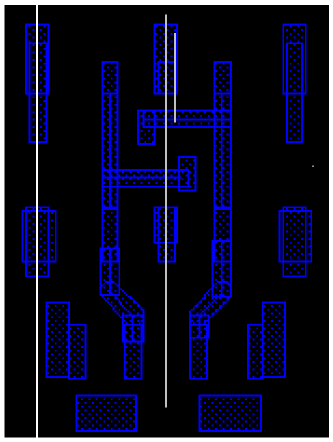
You can control the visibility and selectability of data by layer, color, and color state in the *Layers* panel of the Palette Assistant when *MPT Support* is enabled.

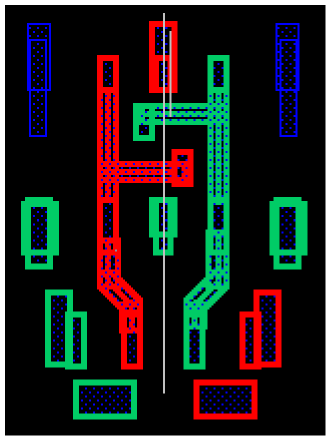


For more information, see Layers Panel in *Virtuoso Layout Suite L User Guide*.

You can also show or hide color globally Using the Multiple Patterning Toolbar, as described in Showing and Hiding Color.




Metal1 without coloring

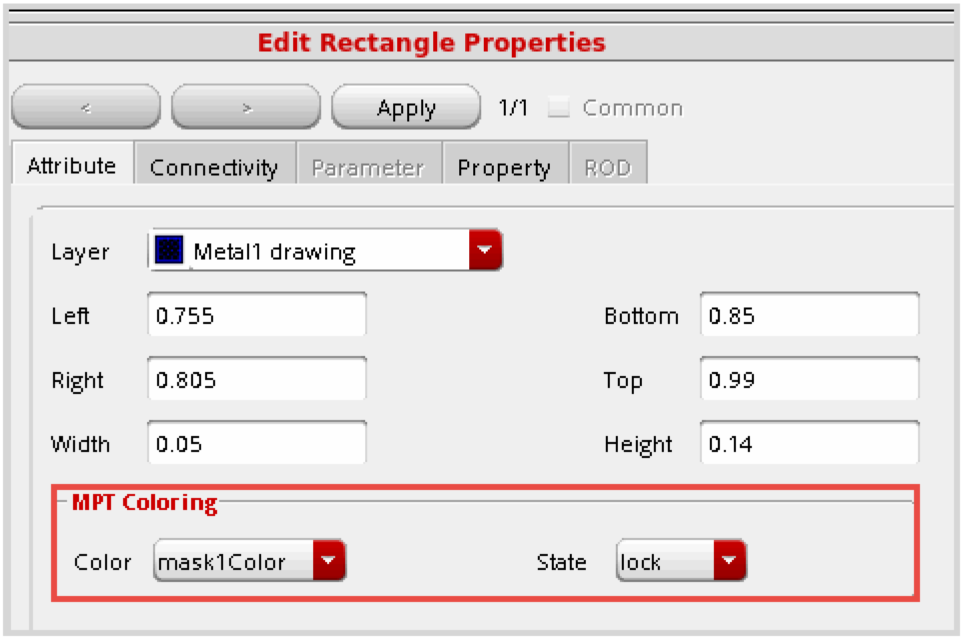

Metal1 with coloring

Inspecting Colored Data

There are several ways to inspect the color and color state of shapes:

- The Show Selection Info Toolbar (*Window – Toolbars – Show Selection Info*) shows the color and color state of preselect and selected objects.

- The Property Editor (*Edit – Basic – Properties* [Q]) shows the color and color state of *selected objects* and *vias* in the canvas in the *MPT Coloring* box of the form.

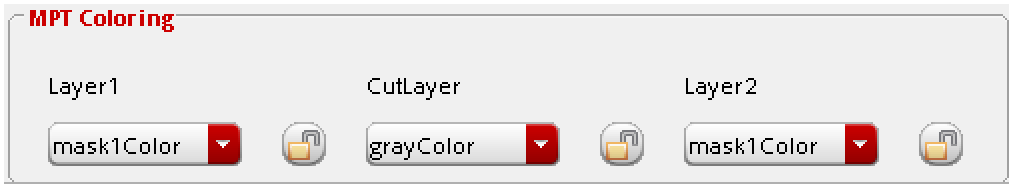

Edit Rectangle Properties

Layer	Metal1 drawing		
Left	0.755	Bottom	0.85
Right	0.805	Top	0.99
Width	0.05	Height	0.14

MPT Coloring

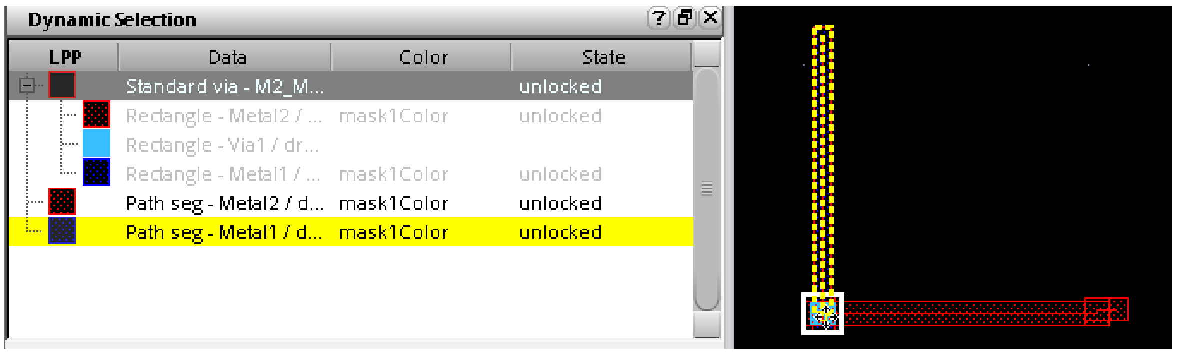
Color	mask1Color	State	lock

For vias, the color and color state (represented by a locked or unlocked icon) for the metal and cut layers are shown.


MPT Coloring

Layer1	CutLayer	Layer2
mask1Color	grayColor	mask1Color

- The Dynamic Selection Assistant (*Window – Assistants – Dynamic Selection*) shows the color and color state for *objects under the cursor* in the canvas. This is useful in densely-populated areas of the canvas.



LPP	Data	Color	State
⬛	Standard via - M2_M...		unlocked
🟥	Rectangle - Metal2 / ...	mask1Color	unlocked
🟦	Rectangle - Via1 / dr...		
🟦	Rectangle - Metal1 / ...	mask1Color	unlocked
🟥	Path seg - Metal2 / d...	mask1Color	unlocked
🟦	Path seg - Metal1 / d...	mask1Color	unlocked

- SKILL functions

  The following SKILL functions query the color and color state of hierarchical (occurrence) shapes:

SKILL Function	Description
`dbColorShapeQuery2`	Returns a list of all shapes in a given region with the color, the color state, and whether the color was set by hierarchical color locking.
`dbGetColoredOccShapes`	Returns a list of colored occurrence shapes with the color, and whether the color is locked for each shape. Only shapes that were colored using hierarchical color locking are included in the list.
`dbGetShapeEffectiveColor`	Returns the color information for an individual occurrence shape. The color information includes the color, and two Boolean values indicating whether the color is locked and whether the shape was colored by hierarchical color locking.

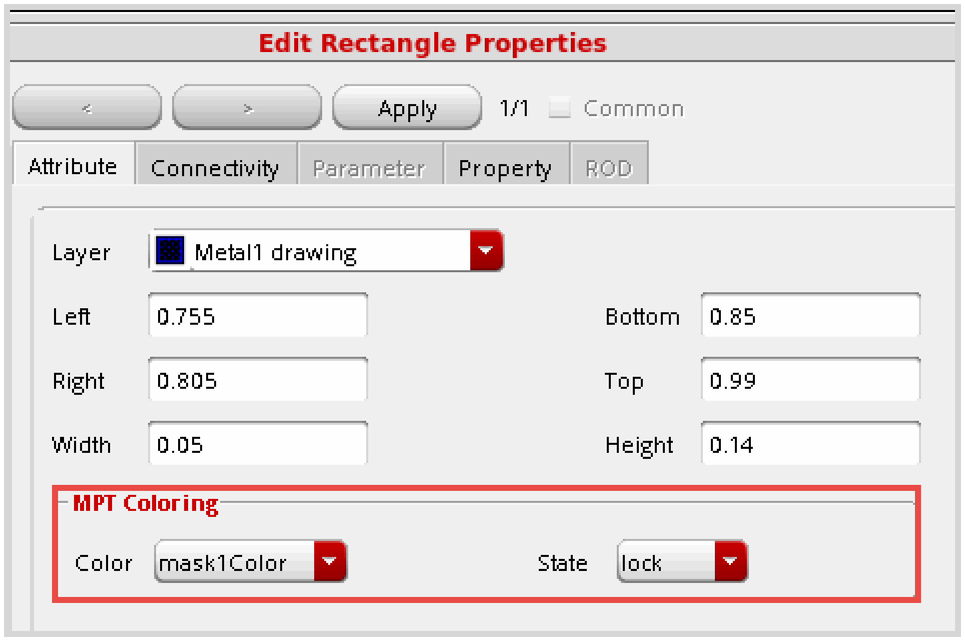
  These SKILL functions can query the color for occurrence shapes:

SKILL Function	Description
`dbColorShapeQuery`	Returns a list of all the shapes in a given region and their effective colors.
`dbGetShapeColor`	Returns the assigned color of the specified shape.

Coloring Existing Shapes

There are several ways to change the color of shapes:

- Use the Property Editor to change the color and color state of selected objects and vias in the canvas.



Layer: Metal1 drawing
Left: 0.755
Bottom: 0.85
Right: 0.805
Top: 0.99
Width: 0.05
Height: 0.14

MPT Coloring
Color: mask1Color
State: lock

- Use the Multiple Patterning Toolbar for the following:
  - Color Shifting shifts the color for unlocked shapes.
  - Color Locking locks and unlocks the color assigned to shapes.
  - Recoloring Selected Colors updates color information for changed data or the entire design.
  - Removing Color from Shapes removes color by layer or for the entire design.
- Use the multiple patterning color engine to color or recolor a design, as described in Using the Multiple Patterning Color Engine.

Color Locking

In layout, there are two types of color locks that can be assigned to a shape:

- The color state for a top-level shape indicates whether the color is locked or unlocked. A `lock` color state is also called a **dbLock**. For more information, see Color Attribute Locking.
- A **hierarchical color lock** (HCL) on an occurrence shape assigns a locked color to the shape from a higher level of the hierarchy. For more information, see **Hierarchical Color Locking.**

By default, both of these methods operate on selected shapes. You can also operate on the selected shapes and shapes that are connected to them on the same layer, as described in Color Locking on Connected Shapes.

Important

New shapes created by functions such as copy, move, and MakeCell, will not inherit the color attributes of the original shape. When shapes are moved to connect to locked shapes, locks are not propagated automatically. To change the default behavior, see Color Locking on Connected Shapes.

In addition, colors can be locked on shapes, vias, and blockages using SKILL functions, as described in Virtuoso SKILL Functions for MPT.

There is a special process for Color Locking Support for Synchronous Clones.

Color Attribute Locking

To lock the color on top-level shapes, you set the color state for the shape to `lock` using one of the following:

- Property Editor, as described in Interactively Setting Color on a Selected Object in *Virtuoso Layout Suite L User Guide*.
- Multiple Patterning toolbar, as described in Locking and Unlocking Colors.

The outline around the shape reflects the assigned color attribute, as shown in Color Representations.

Hierarchical Color Locking

To lock the color of shapes inside an instance, you must use **hierarchical color locking** (HCL). The outline around locked hierarchical shapes is the same as for top-level shapes, as shown in Color Representations. HCL is layer-based, not net-based. You can use the Dynamic Selection Assistant to view the color and color state of shapes under the cursor, as described in Inspecting Colored Data.

Hierarchical color locks are preserved when a Pcell is remastered and when Pcells are abutted. For more information, see Preserving Hierarchical Color Locks on Pcells.

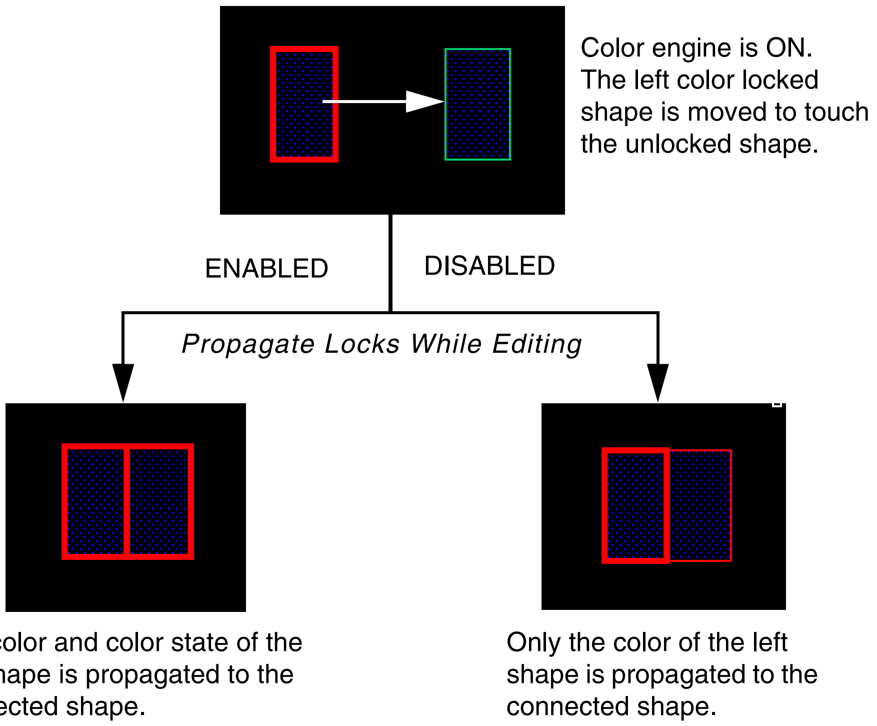
Color Locking on Connected Shapes

By default, color attribute locking and HCL operate only on selected shapes. There are two ways to propagate locks to connected shapes:

- Use *Propagate Locks* in the Multiple Patterning toolbar, as described in Propagating Locks.
- Set the appropriate options in the Multiple Patterning Options form to automatically propagate locks **when the color engine is on**:
    - The following options influence locking that is initiated from the Multiple Patterning toolbar (*Lock Current*, *Lock Colorx, Lock All*):
        - *Propagate Locks to Connected Shapes* extends the color locking to unlocked shapes that are connected to the selected shapes on the same layer*.*
        - *Propagate Locks to Connected Shapes* with *Override Lock on Connected Shapes* extends the color locking to the locked and unlocked shapes that are connected to the selected shapes on the same layer.

      Note: The *Lock Current* option lets you select the instance and locks the current shape. All shapes inside the instance are locked. The values specified by the enableHCLCreation and enableHCLCreationOnPcells environment variables are followed.
    - This option influences locking while editing (for example, when adding, moving, or stretching a shape, or assigning a locked color using the Property Editor):
        - *Propagate Locks During Editing* extends the color locking to unlocked shapes that are connected to the selected shapes on the same layer. The

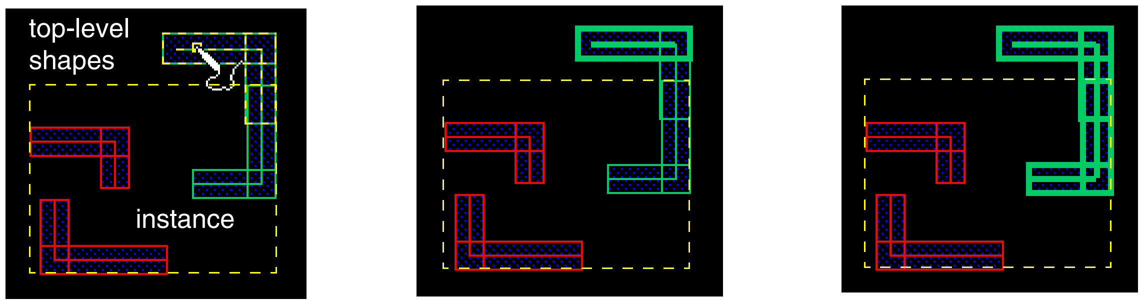
following figure shows the results after a color locked shape is moved to connect to another shape.


Color engine is ON.
The left color locked shape is moved to touch the unlocked shape.
ENABLED
DISABLED
Propagate Locks While Editing
The color and color state of the left shape is propagated to the connected shape.
Only the color of the left shape is propagated to the connected shape.

For more information on setting these options, see Using the Multiple Patterning Options Form.

The difference between hierarchical color locking with *Propagating Locks to Connected Shapes* disabled (default) versus enabled is illustrated in the figure below.

Figure 4-1 Hierarchical Color Locking Default Versus Propagating Locks



a) Two top-level shapes in the upper right corner connect to a shape in the instance. The probe indicates the selection point for b) and c).

b) With HCL **default** mode, only the shapes under the cursor are locked/unlocked.

c) With HCL and *Propagating Locks to Connected Shapes* enabled, the same action locks/ unlocks the shape under the cursor and all shapes (top-level and hierarchical) connected to it on the same layer.

Color Locking Support for Synchronous Clones

To change the color state of a shape in a synchronous clone, you must *Edit In Place* the synchronous clone.

Note: You can delete all colors inside synchronous clones from the top level. It is not mandatory to *Edit In Place* the synchronous clones when deleting all colors.

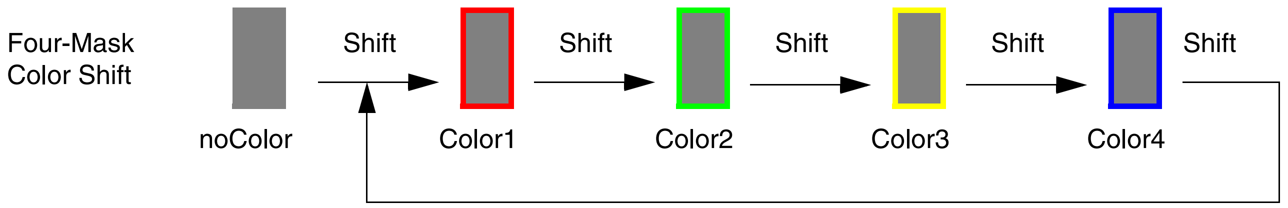
For more information, see Coloring Synchronization Support for Synchronous Clones in *Virtuoso Layout Suite XL User Guide*.

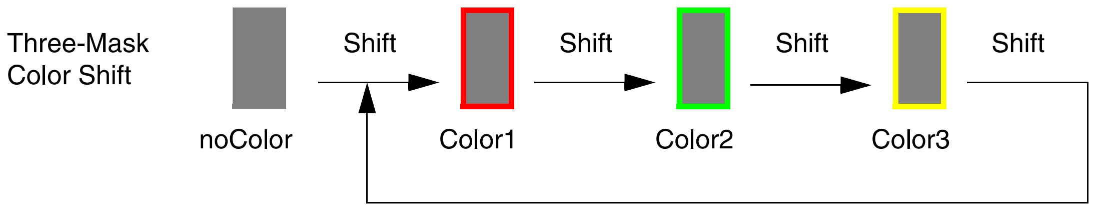
Preserving Hierarchical Color Locks on Pcells

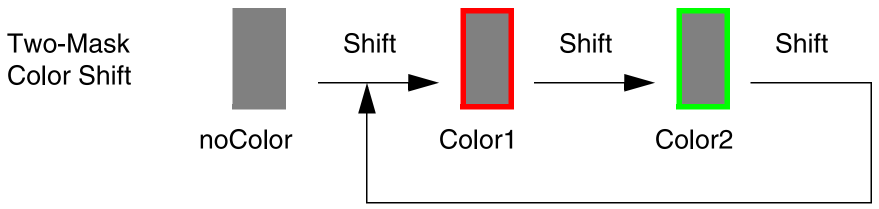
Hierarchical color locks operate on top-level and hierarchical (*occurrence*) shapes and are preserved when a Pcell is remastered and when Pcells are abutted.

Color Shifting

For layers with more than one mask, shapes can be assigned to a specific mask color. The mask for a shape can be manually shifted through the available masks using the Multiple Patterning toolbar, as described in Shifting Colors.







Shifting Colors on Instances

Virtuoso MPT supports *instance* color shifting by layer. The following table lists the layer shift types that are available:

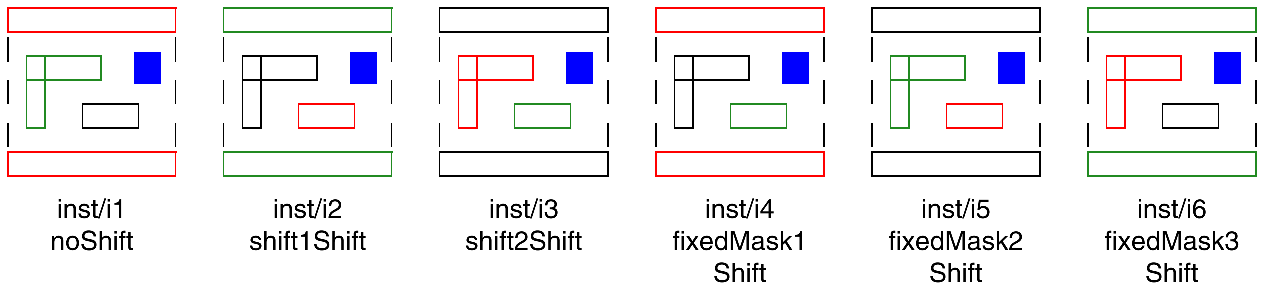
Layer Shift Type	Description
noShift	Colors are not shifted on this layer
shift1Shift	(2-mask and 3-mask layers only) Shift colors by one.
shift2Shift	(3-mask layers only) Shift colors by two.

Layer Shift Type	Description
fixedMask1Shift	(3-mask layers only) Shapes with mask1Color are not changed; shift mask2Color to mask3Color, and mask3Color to mask2Color.
fixedMask2Shift	(3-mask layers only) Shapes with mask2Color are not changed; shift mask1Color to mask3Color, and mask3Color to mask1Color.
fixedMask3Shift	(3-mask layers only) Shapes with mask3Color are not changed; shift mask1Color to mask2Color, and mask2Color to mask1Color.

Gray or uncolored shapes in an instance are not affected by layer shifts on the instance.

The following table shows the effective color for a shape on a three-mask layer based on its original mask color and the layer shift that is applied to the instance.

Original Mask Color	noShift	shift1Shift	shift2Shift	fixedMask1 Shift	fixedMask2 Shift	fixedMask3 Shift
mask1Color	mask1Color	mask2Color	mask3Color	mask1Color	mask3Color	mask2Color
mask2Color	mask2Color	mask3Color	mask1Color	mask3Color	mask2Color	mask1Color
mask3Color	mask3Color	mask1Color	mask2Color	mask2Color	mask1Color	mask3Color



Use the SKILL functions in the following table to initialize, set, retrieve, and clear the layer shift information on a cellview:

SKILL Function	Description
dbCellViewInitForLayerShifting	Initializes the layer shift information for the specified cellview.

SKILL Function	Description
dbCellViewUpdateLayerShifts	Creates or rebuilds the layer shift information cache using the layers that are in the tech bound to the specified cellview. As a result, the layer shift information on the instances in the cellview is removed.
dbCellViewAreLayerShiftsValid	Indicates whether the layer shift information on the instances in the specified cellview is synchronized with the layers and color information in the bound tech.
dbCellViewClearLayerShifts	Clears the layer shift information from all the instances in the specified cellview.
dbCellViewHasLayerShifts	Indicates whether layer shift information is present on any instance in the specified cellview.
dbInstGetLayerShifts2	Returns a list of shifts that are applied to the layers in the master of the specified instance.
dbInstHasLayerShifts	Indicates whether layer color shifts are specified on the instance.
dbInstSetLayerShifts2	Sets layer shifts on an instance.
dbInstClearLayerShifts	Removes all the layer shifting information from the specified instance.

Important

To set the shift type for an instance, you must first initialize the layer shift information for the cellview by using dbCellViewUpdateLayerShifts or dbCellViewInitForLayerShifting.

Example

```
cv = geGetEditCellView()
dbCellviewUpdateLayerShifts( cv )
I2 = dbFindAnyInstByName( cv "I2" )
I3 = dbFindAnyInstByName( cv "I3" )
shiftVals = list(list("Metal2" "fixedMask1Shift") list("Metal3" "shift2Shift"))
dbInstSetLayerShifts2( I2 shiftVals )
dbInstSetLayerShifts2( I3 shiftVals )
```

Sets the layer shift for `Metal2` to `fixedMask1Shift` and for `Metal3` to `shift2Shift` for instances `I2` and `I3` of the current edit cellview.

Pre-defined Setup Driven MPT Flows

You can now easily setup the MPT engine with pre-defined MPT flows. You can use SKILL functions, `mptGetFlowSettings`, `mptGetFlowNames`, `mptSetFlow`, `mptDefineFlow`, `mptCheckFlow`, and `mptReportCurrentSettings` to define the MPT flows.

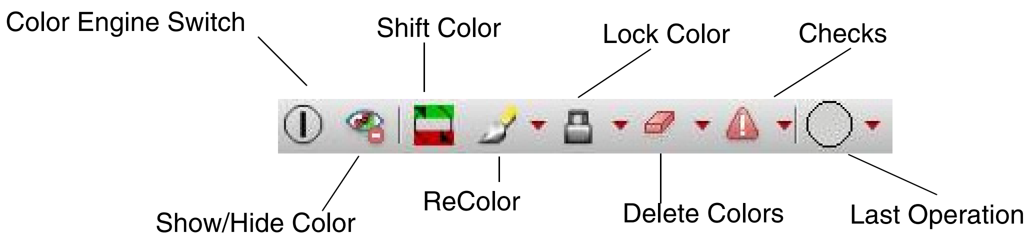
Using the Multiple Patterning Toolbar

To show the Multiple Patterning toolbar, do one of the following:

➡ Choose *Window – Toolbars – Multiple Patterning* from the layout window menu bar.

➡ Access the Multiple Patterning workspace, as described in Using the Multiple Patterning Workspace.

The Multiple Patterning toolbar appears in the toolbar section of the Virtuoso workspace and can be moved in the workspace like other Virtuoso toolbars.



You use the Multiple Patterning toolbar for the following functions:

Table 4-3 Multiple Patterning Toolbar Icons and Functions

Toolbar Icon	Function	For information on this function, refer to:
ⓘ ◯	Color Engine Switch	Turning the Multiple Patterning Color Engine On and Off
👁 👁	Show/Hide Color	Showing and Hiding Color
🟩🟥	Shift Color	Shifting Colors
🖌 ▾	ReColor	Recoloring Selected Colors
🔒 ▾	Lock Color	Locking and Unlocking Colors
▱ ▾	Delete Colors	Removing Color from Shapes
🟥 🟥🟩	Stitch/Unstitch	Using Stitch and UnStitch
🟪	Markers to Mask	Converting Markers to Mask Colors
⚠ ▾	Checks	Checking Colors

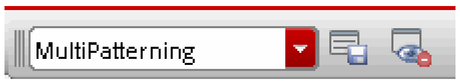
Table 4-3 Multiple Patterning Toolbar Icons and Functions

Toolbar Icon	Function	For information on this function, refer to:
●	Last Operation Status and Color Query	Viewing the Last Operation Status and Probing the Color Source

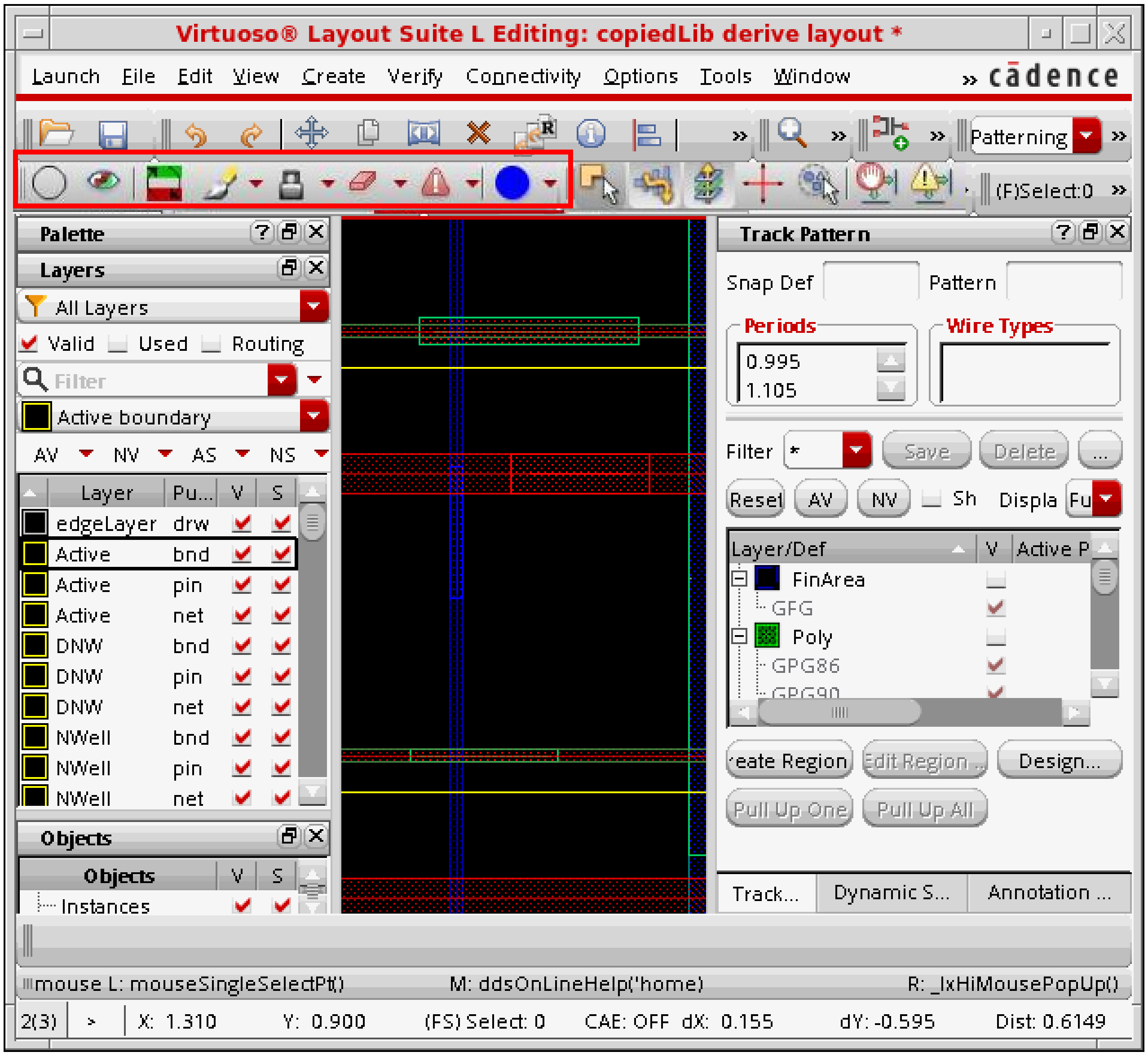
Using the Multiple Patterning Workspace

The Multiple Patterning workspace can be accessed by one of the following methods:

- Choose *Window – Workspaces – MultiPatterning* from the layout window menu bar.
- Choose *MultiPatterning* from the drop-down list box in the Workspaces toolbar.



The layout window is configured with the assistant panes and toolbars that are useful for MPT designs, as shown in the figure below. The Multiple Patterning toolbar is highlighted in red in this figure.



For information on workspaces and how to customize them, see Working with Workspaces in *Virtuoso Design Environment User Guide*.

Showing the Active Multiple Patterning Toolbar Command

The status banner *Cmd* field in the bottom-right corner of the layout window shows the active command for post-select actions initiated from the Multiple Patterning toolbar. For information on these post-select actions, see Post-Select Locking and Unlocking and Removing Color from Shapes.

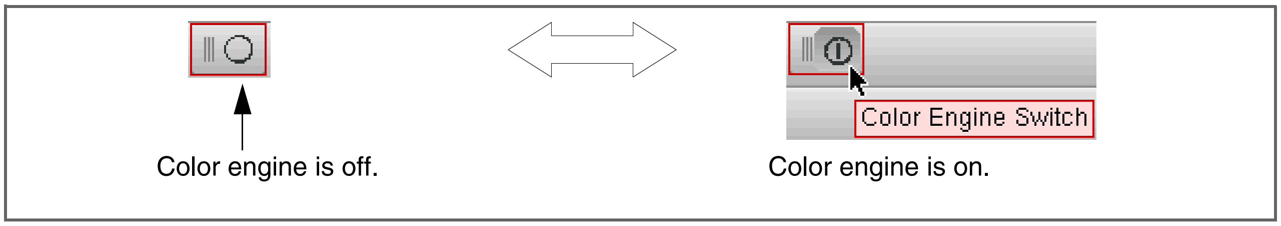
Turning the Multiple Patterning Color Engine On and Off

By default, the multiple patterning color engine is OFF.

When the color engine is off, coloring is not changed when shapes are added, moved, or changed. For more information, see Using the Multiple Patterning Color Engine.

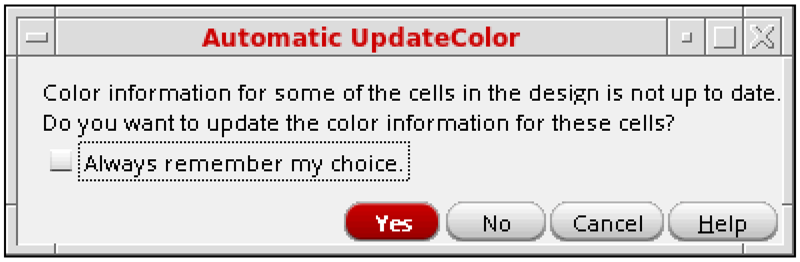
To change the current state of the multiple patterning color engine:

➡ Click the *Engine Switch* icon.



When the Color Engine Is Activated

When the color engine is activated and outdated coloring is detected in the current cellview, the Automatic UpdateColor dialog box appears.


Automatic UpdateColor

Color information for some of the cells in the design is not up to date.
Do you want to update the color information for these cells?

Always remember my choice.

Important

Cadence recommends that you run the color update to ensure that coloring is up to date before further edits are performed. Otherwise, you might see unusual behavior when editing a design with outdated color information.

1. (Optional) Click the *Always remember my choice* checkbox.

   When disabled, the `updateColorOnActivate` environment variable is set to `ask`, which causes this dialog box to appear each time any outdated coloring is detected in the current cellview, and the color engine is activated. When enabled, the `updateColorOnActivate` environment variable is set according to the choice in the next step.

2. Choose one of the following:

   - *Yes*

     Updates colors through the hierarchy. If *Always remember my choice* is enabled, the `updateColorOnActivate` environment variable is set to `always` causing future color updates to occur automatically when the outdated coloring is detected and the color engine is activated.

   - *No*

     Color is not updated. If *Always remember my choice* is enabled, the `updateColorOnActivate` environment variable is set to `never`, You will need to manually update colors, as described in Recoloring Selected Colors.

   - *Cancel*

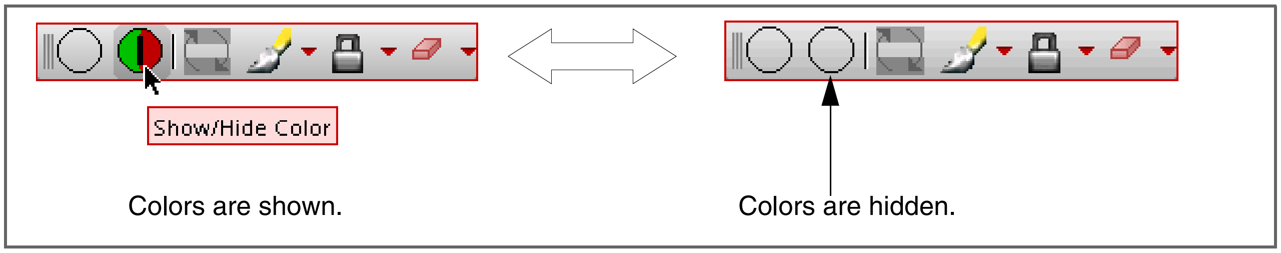
     Color is not updated. The `updateColorOnActivate` environment variable is set to `ask`.

Showing and Hiding Color

By default, mask colors are displayed on visible layer-purposes.

To change the show/hide color setting:

➡ Click the *Show/Hide Color* icon.



Shifting Colors

To change the color of a shape or a set of shapes:

1. Select the shapes in the layout.
2. Click the *Shift Colors* icon.

   Shapes on layers with more than one mask will shift through colors, as described in Color Shifting.

For information on color shifting instances, see Shifting Colors on Instances.

Color Shifting Locked Shapes

By default, color shifting is not permitted on locked shapes. There are multiple patterning options for changing this behavior:

- *Allow Shifting of Locked Shapes* permits color shifting on individual locked shapes.
- *Allow Shifting of Locked Shapes* with *Override Lock on Connected Shapes* and *Propagate Locks to Connected Shapes* permit color shifting on connected locked shapes on the same layer.

For more information on setting these options before shifting colors, see Using the Multiple Patterning Options Form.

Recoloring Selected Colors

When you first open a design that has not been colored, or for which the color information is out of date, use one of the following methods to update color:

- Recoloring Selected Shapes

  Use this to recolor specific shapes (for example, shapes in a region).

- *Recoloring Visible Area*

  Use this method to recolor the shapes in the visible area.

- *Recolor All*

  Use this method when you load a design that has never been colored, is out of date, or was colored using an early software version. By default, designs are not automatically colored when opened.

Important

For all these methods, the resulting color is dependent on the current coloring method (interactive or managed), and the environment variable settings. The color engine does not need to be ON to update color using these methods.

To determine whether the color information is out of date, see Checking the Color Status of a Design.

Coloring can be slow for large designs. You can interrupt the coloring process by pressing the `Ctrl` + `C` keys.

Important

When coloring is interrupted, the data is left in an unknown state.

Checking the Color Status of a Design

There are two ways to check the color status of a design:

- Finding the outdated cellviews

  Use the mptGetOutdatedDesigns SKILL function to get a list of the cellviews in the hierarchy for which the coloring is outdated. You can optionally include the outdated layers for each cellview in the returned list.

- Finding the up-to-date cellviews

  Use the mptGetUpToDateDesigns SKILL function to get a list of the cellviews in the hierarchy for which the coloring is up to date.

Controlling the Recoloring of Read-only Cells and Pcells

When you are recoloring or updating color, you must consider how read-only cells and Pcells are treated. Although these can be recolored, the coloring cannot be saved.

- If your design contains read-only cellviews:
    - If there are read-only cellviews that are not fully colored, enable the Recolor readOnly cellviews option (this is the default setting) to permit recoloring of the read-only cellviews in memory. This will determine the colorability of the design, but the coloring in memory for the read-only cellviews cannot be saved.
    - If the read-only cellviews are fully colored, you can disable the Recolor readOnly cellviews option. The existing color in the read-only cellviews will be used but those cellviews will not be recolored. This can reduce the recoloring processing time.
    - If you wish to save the coloring for the read-only cellviews, you must first make them editable. You can make them editable in step 3 of the *Recolor All* procedure below.
- If your design contains Pcells:
    - By default, when the Don't color Pcells option is disabled, Pcell shapes will be recolored in memory. This will determine the colorability of the design, but the coloring for the Pcells cannot be saved.
    - If you fully color the Pcell shapes directly in the Pcell code or using CDF parameters, you should enable the Don't color Pcells option. Existing color in the Pcells will be recognized but the Pcells will not be recolored, thus reducing the recoloring processing time.

    Note: To save Pcell coloring in the design, you must either use express Pcells or apply Hierarchical Color Locking to save the coloring at the instance level.

For more information on setting these options, see Using the Multiple Patterning Options Form.

Recoloring Selected Shapes

You can update the color of selected shapes by using the *ReColor Selected* option. To update the color of selected shapes, use the `mptReColorFromShapes` SKILL function. The color of the shapes connected to the shapes in the list will also be updated. Additionally, if the coloring method is `managed`, then the color of the shapes that are within the same-mask spacing of the shapes in the list will be updated, according to the clustering algorithm described in Automatic Color Shifting.

Recoloring Visible Area

You can update the color of shapes in the visible area using the *ReColor Visible Area* option.

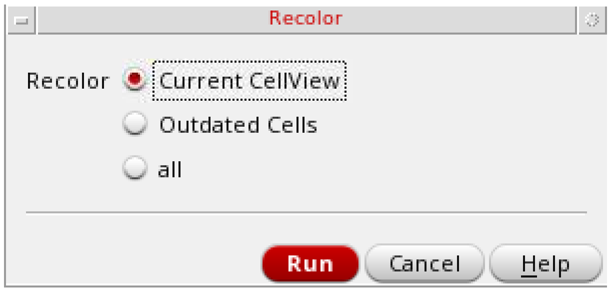
Recolor All

By default, *Recolor All* will clear all colors, then color the entire design. This is useful for determining if the design is colorable. If your design contains read-only cellviews or Pcells, refer to Controlling the Recoloring of Read-only Cells and Pcells to ensure that these are handled appropriately.

To recolor all data:

1. Choose *ReColor All* from the *Recolor Selected* drop-down list.

   The Recolor form appears.



Recolor
Recolor [✓ Current CellView] Outdated Cells all

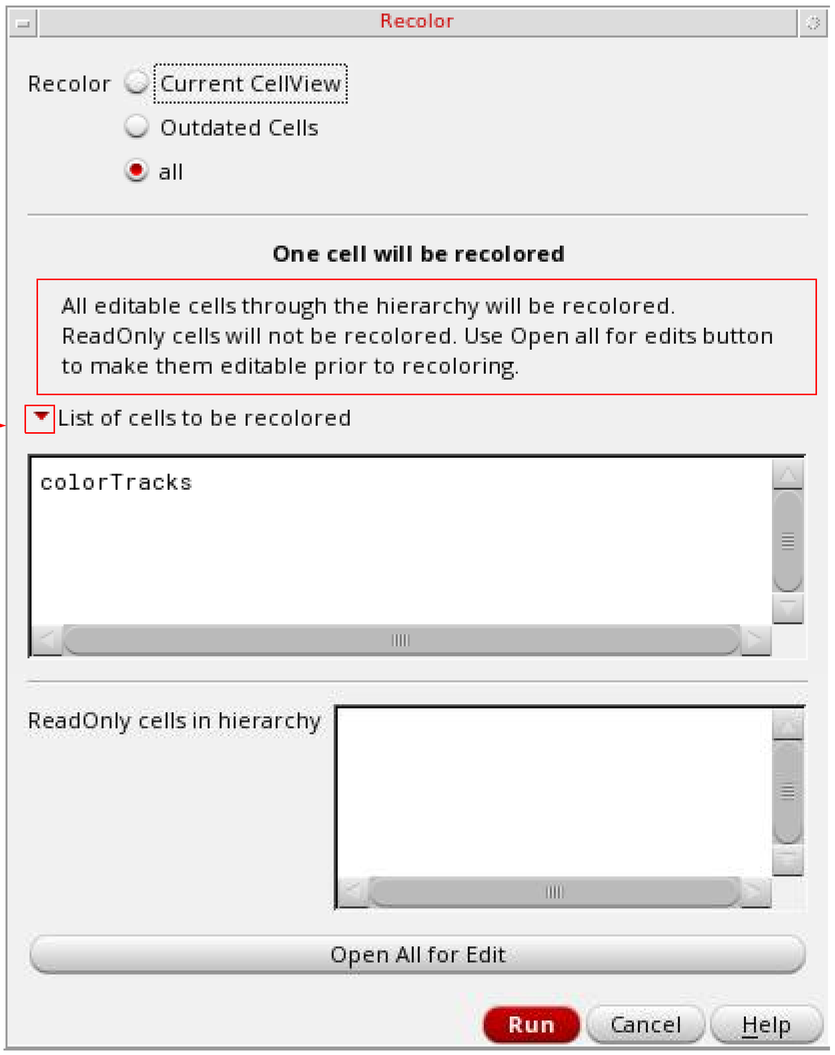
2. You can select *Current CellView*, *Outdated Cells*, or *all*.

   When you select the *Current CellView* option, the shapes in the current cellview are recolored.

   The *Outdated Cells* option, instantiates an uncolored cell in a design that is already colored. This method runs faster than the *all* option, for cases when only part of the data is out of date because valid color information is not recomputed.

   The *all* option recolors all the cells.

   You can use the `reColorGUIScope` environment variable to set and query the *ReColor* options.


Recolor

Recolor ○ Current CellView
○ Outdated Cells
● all

One cell will be recolored

All editable cells through the hierarchy will be recolored.
ReadOnly cells will not be recolored. Use Open all for edits button to make them editable prior to recoloring.

List of cells to be recolored

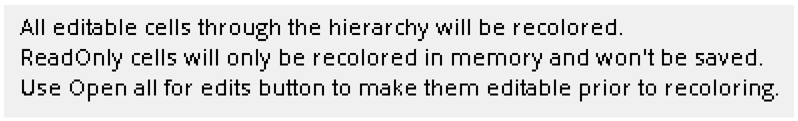
```
colorTracks
```

ReadOnly cells in hierarchy

Advisory: *Recolor readOnly Cellviews* option is not enabled

Click to expand/ collapse the list of cells

When the *Recolor readOnly Cellviews* option is enabled, the default advisory shows the following:


All editable cells through the hierarchy will be recolored.
ReadOnly cells will only be recolored in memory and won't be saved.
Use Open all for edits button to make them editable prior to recoloring.

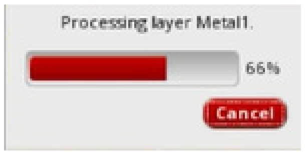
3. (Optional) If there are read-only cells in the hierarchy for which you want to save the coloring, click *Open All for Edit* to make them editable.

4. Click *Run*.

The design is recolored based on the coloring method setting. See Checking for Multiple Patterning Violations to determine if there are any colorability issues.

The equivalent SKILL function is `mptReColor`.

A progress bar appears if the *Recolor All* option or `mptReColor` option takes longer than a few seconds to complete. This provides a view of the progress of the recoloring task. It also gives you the choice to cancel the recolor command.


Processing layer Metal1.

66%

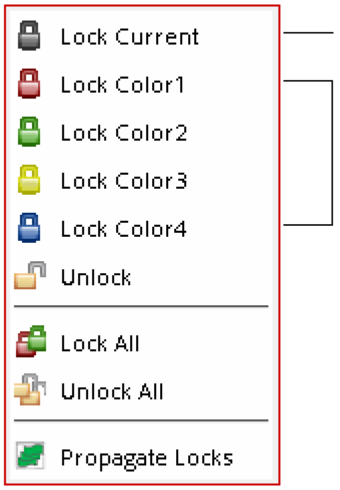
Locking and Unlocking Colors

When using the Multiple Patterning toolbar, the type of color locking performed is dependent on whether the shapes have already been selected.

To lock or unlock the color on a shape or a set of shapes, or the entire design in the layout:

1. Click the *Lock Colors* icon in the Multiple Patterning toolbar.

2. Choose the desired action from the drop-down list.



Locks to the current color.

Locks to the specified color. Available colors are based on the number of masks defined in the technology file for the layer.

Action	**Shapes are preselected**	**Shapes are NOT preselected**
Lock Current *Lock Color1* *Lock Color2* *Lock Color3* *Lock Color4*	Locks the selected shapes to the specified color using Color Attribute Locking.	Locks the selected shapes to the specified color as shapes are selected. For information, refer to Post-Select Locking and Unlocking.
Unlock	Sets the color state for the selected shapes to `unlock`.	Unlocks the color of shapes as shapes are selected. For information, refer to Post-Select Locking and Unlocking.
Lock all	Locks all the colored shapes at the top level or at the current editing hierarchy level (top level), as described in Lock and Unlock All.	
Unlock all	Unlocks all the colored shapes at the top level or at the current editing hierarchy level (top level), as described in Lock and Unlock All.	

Action	Shapes are preselected	Shapes are NOT preselected
Propagate Locks	Propagates locks for connected shapes, same color groups, and net-based pre-colored shapes, as described in Propagating Locks. You can also propagate locks automatically when the color engine is on and you set certain Multiple Patterning Options, as described in Color Locking on Connected Shapes.	

Locked shapes have a bold outline. For a description of the colors and color styles used to indicate locking, refer to Color Representations. For more information on color locking, refer to Color Locking.

Post-Select Locking and Unlocking

If no shapes were preselected, the cursor becomes a *probe* in the layout window, indicating that *post-select* mode is active. The status banner *Cmd* field at the bottom-right corner of the window shows the active command (*Lock Current, Lock Color1, Lock Color2, Lock Color3, Lock Color4*, or *Unlock*). Click shapes in the canvas to lock or unlock them.

- Lock

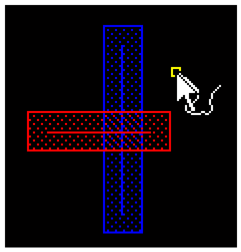
  Shapes at the current editing hierarchy level are locked by setting their color state. Shapes at a lower level of the hierarchy are set with a hierarchical color lock at the current level except when a lock is already set at the lower level. You can also prevent hierarchical color locking on sub-level shapes by setting the `enableHCLCreation` environment variable to `nil`.

- Unlock

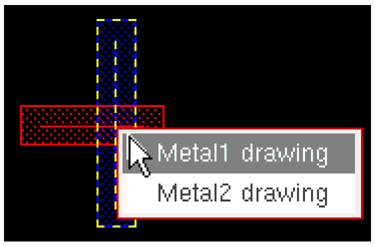
  Color attribute locks and hierarchical color locks at the current editing hierarchy level will be removed. If a shape is locked at a different level of the hierarchy, it will remain locked.

If visible and active shapes on more than one colorable layer exist under the probe, a pop-up menu appears. Choose the layer-purpose pair for the shape that you want locked or unlocked. Figure 4-2 shows an example of post-select locking in the layout.

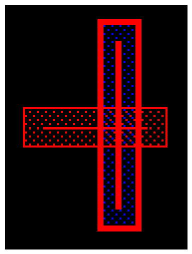
Figure 4-2 Example of Post-Select Locking with Shapes on Multiple Colorable Layers



a) With no shapes selected, *Lock Color1* is chosen in the Multiple Patterning toolbar. The cursor changes to a probe.



b) The probe is clicked on a location with overlapping Metal1 and Metal2 shapes. Since both are colorable layers, a pop-up menu appears with the layer-purpose pairs for the shapes. In this example, Metal1 drawing is selected.



c) The vertical Metal1 drawing shape is locked to Color1.

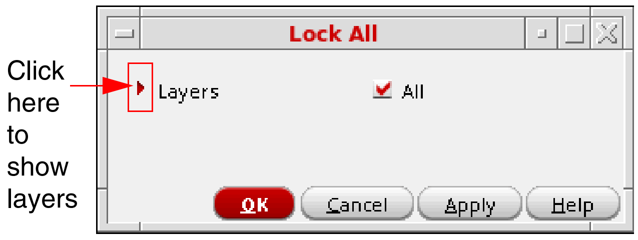
Lock and Unlock All

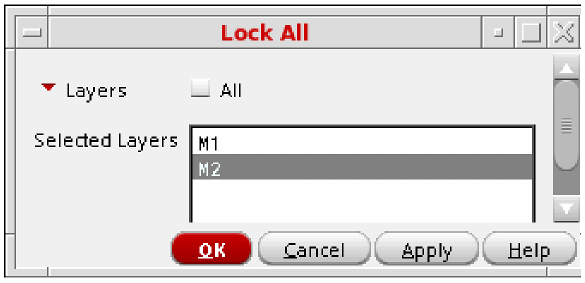
You can lock and unlock all colors for all layers or only specified layers.

To lock or unlock all colors,

1. Choose *Lock All* or *Unlock All* in the *Lock Colors* drop-down list of the Multiple Patterning toolbar.

   The Lock All or Unlock All form appears.





2. (Optional) To lock or unlock colors only on specific layers:

   a. Disable *All*.

   b. Click the expansion button to show the *Selected Layers* list, if it is not visible.

   c. Choose the layers.

3. Click *OK* or *Apply*.

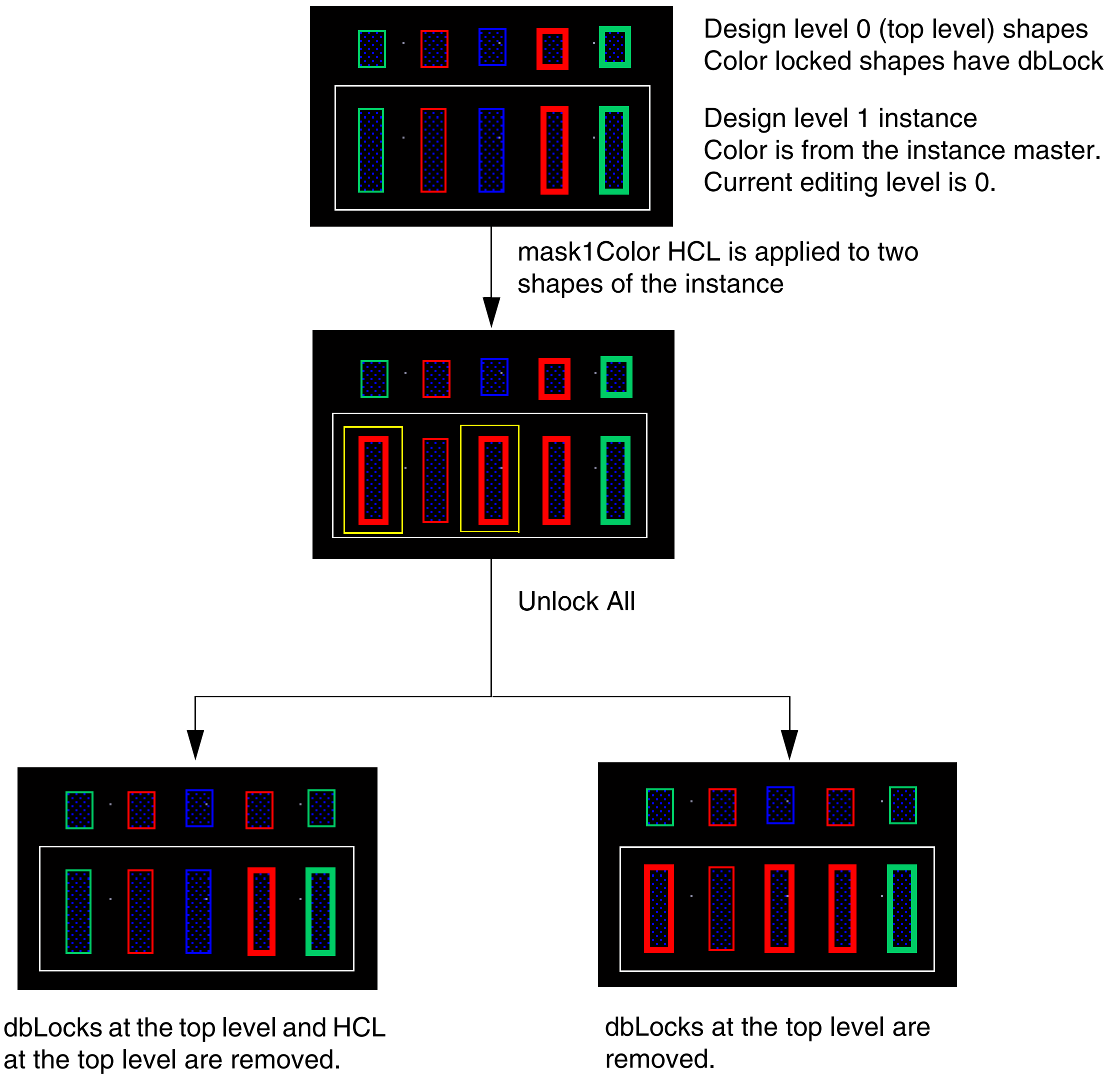
For *Lock All*, uncolored shapes cannot be locked.

Note: The *Lock All* option locks all the shapes inside the synchronous clones from the top-level. While locking, the colors are also modified so that all the synchronous clone colors are synchronized.

For *Unlock All*, top-level shapes with color attribute locks (`dbLocks`) become unlocked with their current color. Hierarchical color locks (HCLs) at the current editing hierarchy level will be removed.

Note: If *Unlock All* option is selected, the values specified for the environment variables, lockAllHCLPolicy, enableHCLCreation, and enableHCLCreationOnPcells are ignored and top-level shapes with color attribute locks become unlocked with their current color.

The following example, mask1Color HCL is applied to the first and third shape of the instance, followed by *Unlock All*.


Design level 0 (top level) shapes
Color locked shapes have dbLock
Design level 1 instance
Color is from the instance master.
Current editing level is 0.
mask1Color HCL is applied to two shapes of the instance
Unlock All
dbLocks at the top level and HCL at the top level are removed.
dbLocks at the top level are removed.

You can also use the `mptLockAll` and `mptUnlockAll` SKILL functions to lock and unlock colored shapes.

Propagating Locks

Lock propagation does the following:

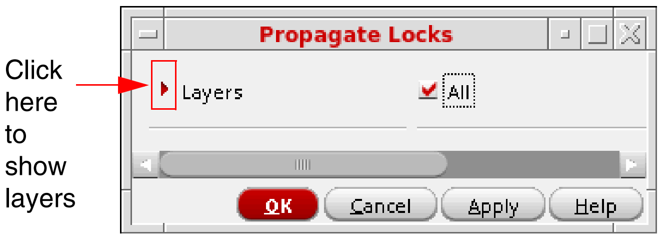
- Color locked shapes are identified. Then, all the shapes connected to the locked shapes on the same layer are locked to the same color throughout the hierarchy, from the current to the bottom level.
- All shapes in critical nets that were pre-colored in schematic, as described in Net-Based Pre-Coloring Flow, will be locked to their current colors.

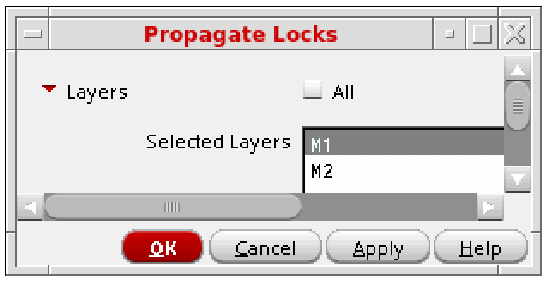
By default, lock propagation operates on all layers, but can be limited to specific layers.

To propagate locks:

1. Choose *Propagate Locks* in the *Lock Colors* drop-down list of the Multiple Patterning toolbar.

   The Propagate Locks form appears.





2. (Optional) To propagate locks only on specific layers:
   a. Disable *All*.
   b. Click the expansion button to show the *Selected Layers* list if it is not visible.
   c. Choose the layers.
3. Click *OK* or *Apply*.

You can also use the `mptPropagateLocks` SKILL function to propagate locks in the entire hierarchy.

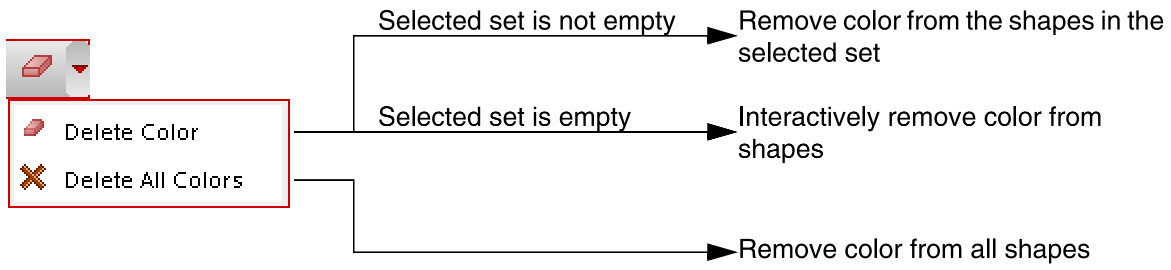
Video

For an overview of this feature, see Lock Propagation.

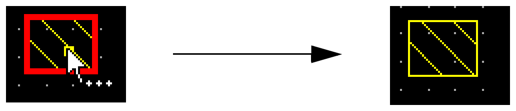
Removing Color from Shapes

To remove color from shapes in the layout:

1. Click the *Delete Colors* icon on the Multiple Patterning toolbar.
2. Choose one of the following from the drop-down list:


Delete Color
Delete All Colors
Selected set is not empty
Remove color from the shapes in the selected set
Selected set is empty
Interactively remove color from shapes
Remove color from all shapes

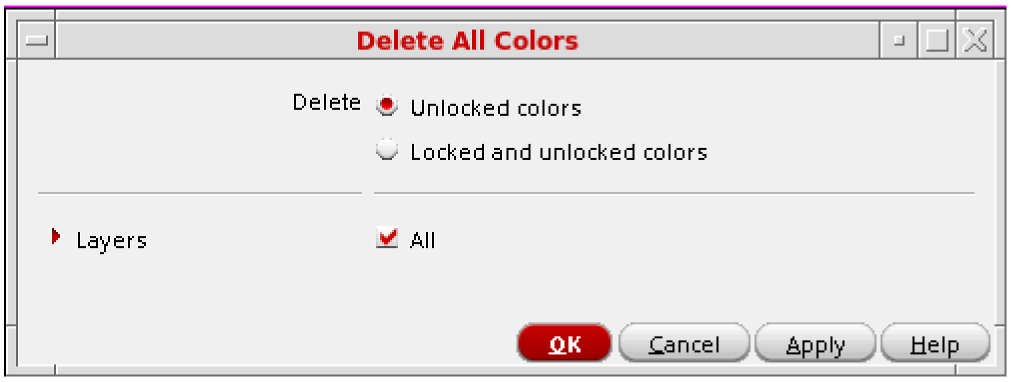
- *Delete Color*
  - If the selected set is not empty, removes the locked and unlocked colors from the shapes in the selected set, including HCL-colored shapes in the selected instances.
  - If there are no shapes selected in the layout, the remove color cursor (arrow with three dots) appears when the layout window is active. The status banner *Cmd* field at the bottom-right corner of the window shows *Delete Color*. Click shapes in the current editing hierarchy level to remove their color.



Note: In the post-selection mode, when you select the *Delete Colors* option, if a layer is not visible in the Palette, the color of the shapes of this layer are not deleted.

- *Delete All Colors*

The Delete All Colors form appears.



Delete All Colors

Delete [✓ Unlocked colors] Locked and unlocked colors

Layers [✓ All]

Using the Delete All Colors Form

In the Delete All Colors form:

1. Choose one of the following:

   ❑ *Delete – Unlocked colors*

   Removes the unlocked colors from shapes and vias at the current editing hierarchy level, and from instance shapes that were colored using hierarchical color locking (HCL).

   ❑ *Delete – Locked and unlocked colors*

   Removes locked and unlocked colors from shapes and vias at the current editing hierarchy level, and from instance shapes that were colored using HCL.

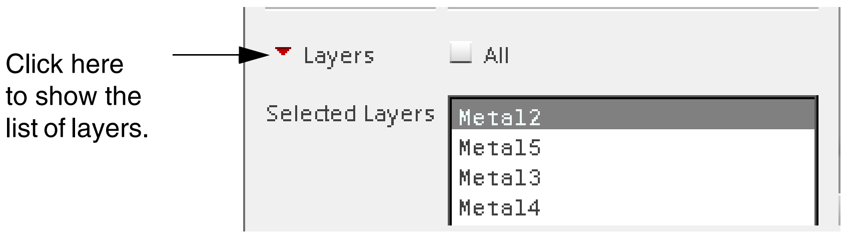
2. Choose the layers for color removal.

   ❑ *All*

   Removes color from all layers.

   ❑ *Selected Layers*

   Turn off *All* and show the list of *Selected Layers* by clicking the expand icon.



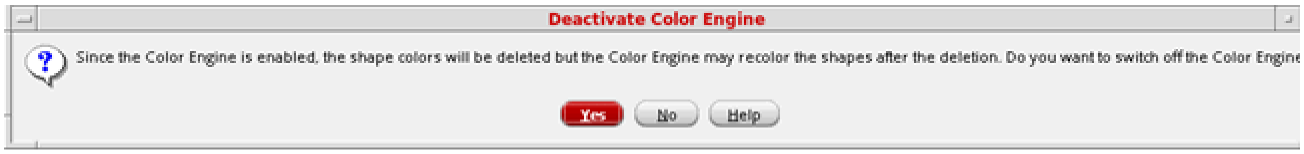
   Select the required layer from the *Selected Layers* list. Click and drag to select more than one consecutive layer. Hold down the `Ctrl` key and click to select/deselect a layer without changing the selection of the other layers. Hold down the `Shift` key and click to select all the layers between, including the last selected layer and the currently selected layer.

3. Click *OK* or *Apply*.

   The colors are removed from shapes as specified by the Delete All Colors form selections.

Turning Off the Color Engine When Removing Colors

If the color engine is on when you are Removing Color from Shapes, the shapes can be recolored by the color engine after the color is removed. When this condition exists, the Deactivate Color Engine dialog appears (before color is removed) that lets you turn off the color engine to prevent recoloring.


Deactivate Color Engine

Since the Color Engine is enabled, the shape colors will be deleted but the Color Engine may recolor the shapes after the deletion. Do you want to switch off the Color Engine

➡ Click *Yes* to turn off the color engine and prevent recoloring, or *No* to keep the color engine on.

Using Stitch and UnStitch

Stitch is a method that can be used to fix color conflicts by breaking one shape into two overlapping “stitched” shapes, each assigned to a different mask.

UnStitch is used to remove stitches.

	Enable Stitch mode.
	Enable UnStitch mode.

Note: By default, the *Stitch* toolbar is hidden. Some processes do not support stitching. If your process supports stitching, you can show the *Stitch* toolbar by setting the `hideStitchingTools` environment variable to `nil`.

For detailed information on how to use stitch mode, refer to Using Stitch for Multiple Patterning Violations.

Converting Markers to Mask Colors

Some imported stream files create duplicate shapes to represent coloring, one for the shape and the other for the color marker shape. Before using the color engine, the imported data of this type must be converted using the `mptMarkersToMaskColors` SKILL function. The Multiple Patterning toolbar also provides a convenient way to perform this conversion.

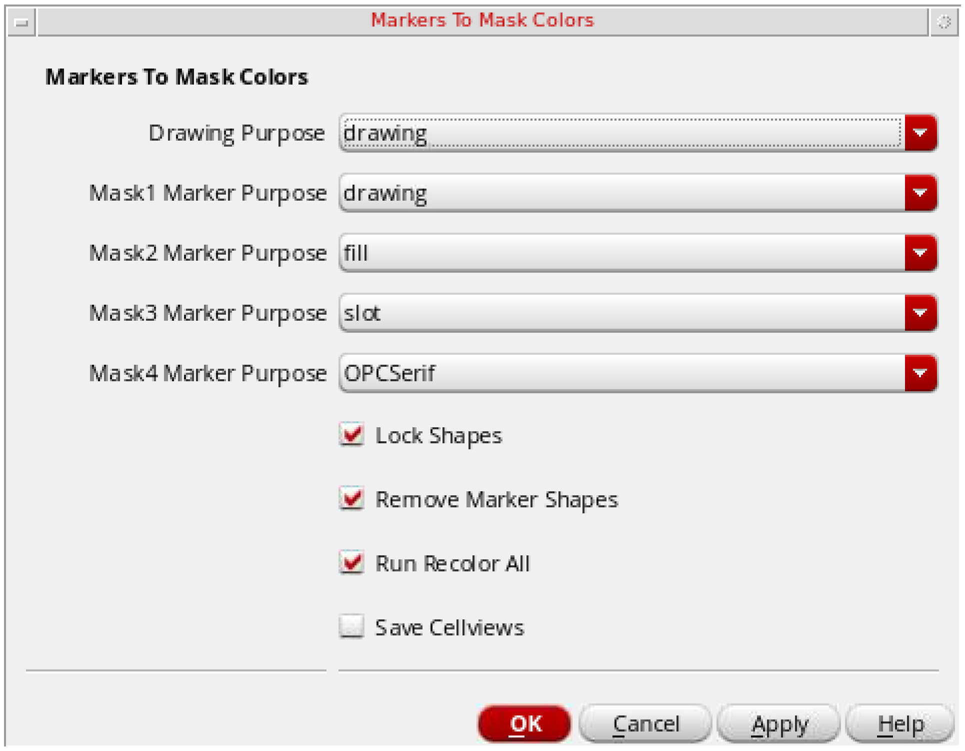
Note: By default, the *Markers to Mask* icon is not visible in the Multiple Patterning toolbar. To display the icon, you must set the `enableMarkersToMaskColors` environment variable:

```
envSetVal("mpt" "enableMarkersToMaskColors" 'boolean t)
```

To convert the color markers to mask colors:

1. Click the *Markers to Mask* icon.

   The Markers to Mask Colors form appears.



Markers To Mask Colors

Markers To Mask Colors

Drawing Purpose	drawing
Mask1 Marker Purpose	drawing
Mask2 Marker Purpose	fill
Mask3 Marker Purpose	slot
Mask4 Marker Purpose	OPCSerif

- ✓ Lock Shapes
- ✓ Remove Marker Shapes
- ✓ Run Recolor All
- ☐ Save Cellviews

2. Choose the *Drawing Purpose*.
3. Choose the *Mask1 Marker Purpose*.
4. Choose the *Mask2 Marker Purpose.*
5. Choose the *Mask3 Marker Purpose.*
6. Choose the *Mask4 Marker Purpose.*

   The purposes apply only to layers with more than two mask colors.
7. Select *Lock Shapes* to lock the color state on shapes.
8. Select *Remove Marker Shapes* to remove the marker purpose shapes.

9. Select *Run Recolor All* to recolor all cellviews.
10. Select *Save Cellviews* to automatically save the modified cellviews in the hierarchy.
11. Click *OK*.

Shapes are merged from the drawing purpose and the marker purposes through the hierarchy so that the shapes on the drawing purpose remain with the expected mask color assignment. You can also do the conversion using `mptMarkersToMaskColors`, described in *Virtuoso Layout Suite SKILL Reference*.

Checking Colors

You can use the options in the *Checks* drop-down list to perform color checks in the design. You can use the following options to check for potential color conflicts in the design:

- Hierarchical Color Locking Check
- Violation Checks
- Methodology Compliance

Hierarchical Color Locking Check

As mentioned in Color Locking section, there are two types of color locks that can be assigned to a shape: the color state (dbLock) and hierarchical color locks (HCL).

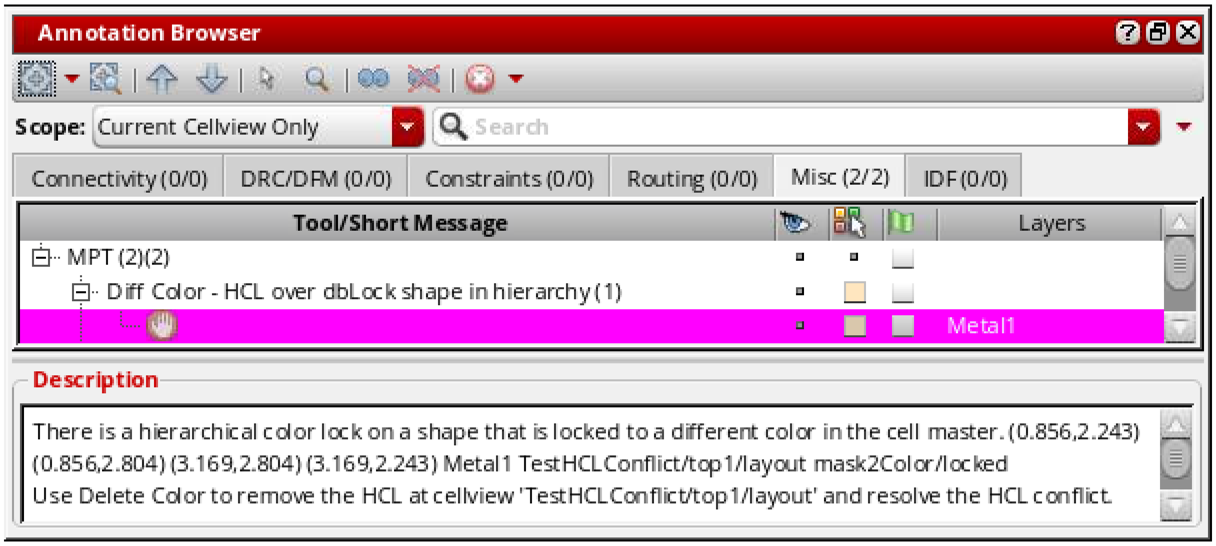
While a shape can have only one dbLock, it could potentially have multiple HCLs applied from various levels of the hierarchy. A color conflict exists when there are multiple locked color assignments for a shape.

To check for potential color conflicts related to color locking:

1. Choose *Hierarchical Color Locking Check* from the *Checks* drop-down list.

   The number of conflicting coloring locks is reported in the CIW. Annotation markers will appear in the canvas where the conflicts were found.

2. Choose *Window – Assistants – Annotation Browser* to open the Annotation Browser and inspect the conflicts listed on the *Misc* tab in the *MPT* grouping.



There is a hierarchical color lock on a shape that is locked to a different color in the cell master. (0.856,2.243) (0.856,2.804) (3.169,2.804) (3.169,2.243) Metal1 TestHCLConflict/top1/layout mask2Color/locked
Use Delete Color to remove the HCL at cellview 'TestHCLConflict/top1/layout' and resolve the HCL conflict.

Potential conflicts include:

❑ *HCL in hierarchy over HCL in hierarchy*

  There are hierarchical color locks of different/same colors at different levels of the hierarchy for a shape.

❑ *HCL over dbLock shape in hierarchy*

  There is a hierarchical color lock on a shape that is also locked in the cell master.

❑ *HCL over shape in locked via*

  There is a hierarchical color lock on a shape that is part of a locked or partially locked via.

3. Resolve conflicts by Removing Color from Shapes.

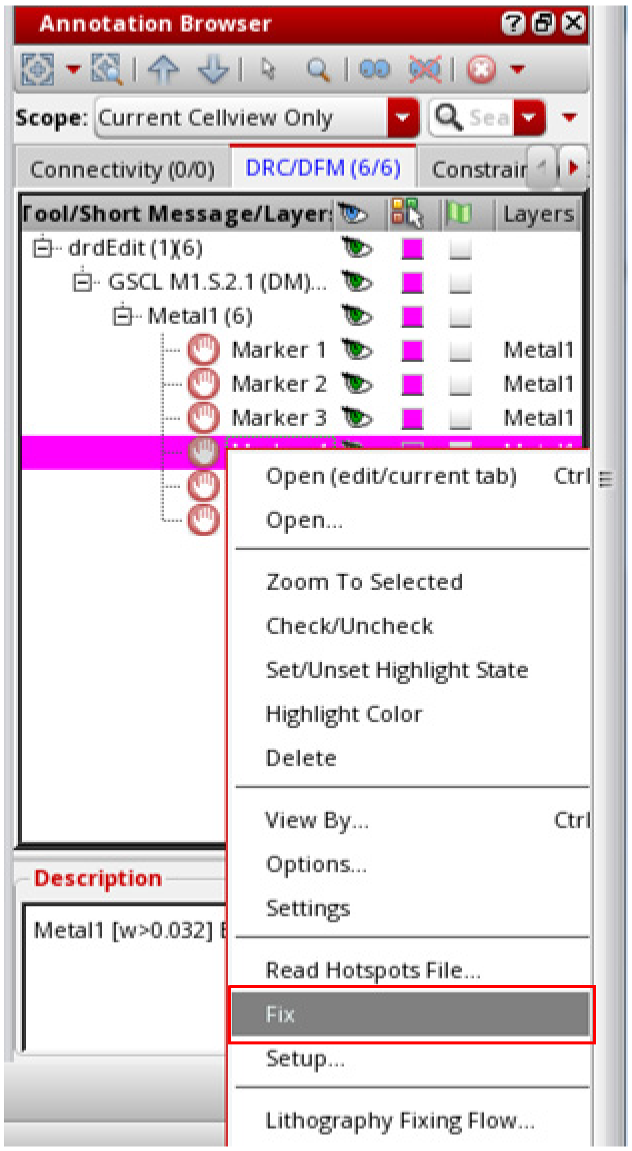
Resolving HCL Conflicts through the Annotation Browser

You can resolve HCL conflicts using Annotation Browser. You can resolve the errors one at a time or as a group.

To resolve HCL conflicts:

1. Select the conflict or group of conflicts in the Annotation Browser.

2. Select *Fix* from the context-sensitive menu.



After the fix, HCLs at the current editing level are cleared and the marker is deleted. For HCLs inside the hierarchy, a message is printed in the CIW specifying the resolution of the error. However, in this case, the marker is not deleted.

Video

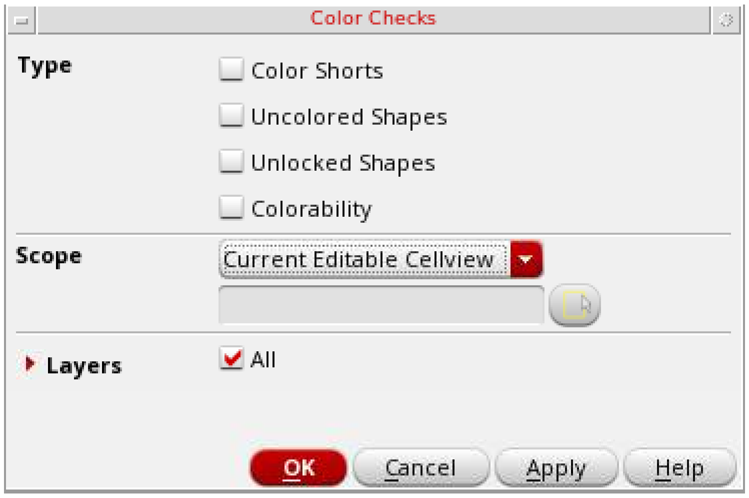
For a video overview of this feature, see Using the Hierarchical Color Locking Check on Cadence Online Support.

Violation Checks

You can perform color checks using the *Checks* option on the Multiple Patterning toolbar.

To perform color checks:

1. Click the *Checks* icon or choose *Violation Checks* from the *Checks* drop-down list.



2. Select the *Type* of color checks that you need to perform. You can select any or all the options: *Color Shorts*, *Uncolored Shapes*, *Unlocked Shapes*, or *Colorability*.

   Note: The *Colorability* option checks if the objects in the design can be colored.

3. Choose the *Scope: Current Editable Cellview*, *Current Visible Area*, or *Area*. The *Current Editable Cellview* checks the current cellview. This option is selected by default. The *Current Visible Area* checks area of the design that is currently visible. The *Area* checks the area that you select in the design.

4. Select the *Layers: All* or *Selected Layers*.

The Verify Process Rules Summary is displayed in the CIW.

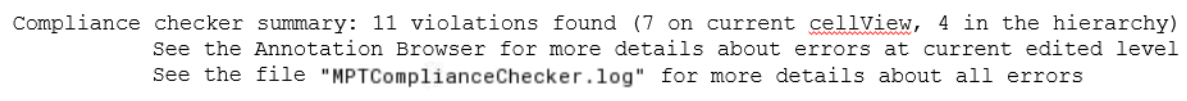
Methodology Compliance

The methodology compliance checker lets you verify that your design matches the desired coloring flow. It flags discrepancies between the design data and the multiple patterning flow setup. For example, it flags shapes having coloring information whereas based on the MPT setup they should have been not colorable.

The methodology compliance checker checks for the following:

- Colored LPPs: Flags any shape that is colored even if it is not part of the colored LPPs.
- Hierarchical Color Locks: The hierarchical color locks are checked using the environment variables, enableHCLCreation, enableHCLCreationOnPcells, globalColorShiftingPolicy, and globalColorShiftingPolicyForPcells.

A short summary of the discrepancies is reported in the CIW. A report is also created with details about each error at the current hierarchy level and inside the hierarchy. A marker is also created for each current level error. You can also view the marker in the Annotation Browser.



```
Compliance checker summary: 11 violations found (7 on current cellView, 4 in the hierarchy)
           See the Annotation Browser for more details about errors at current edited level
           See the file "MPTComplianceChecker.log" for more details about all errors
```

You can use the environment variable complianceCheckerReport to specify the file name and location of the methodology compliance checker report. The environment variable, complianceCheckerLimit, is used to specify the maximum number of violations that are reported by the methodology compliance checker.

Viewing the Last Operation Status and Probing the Color Source

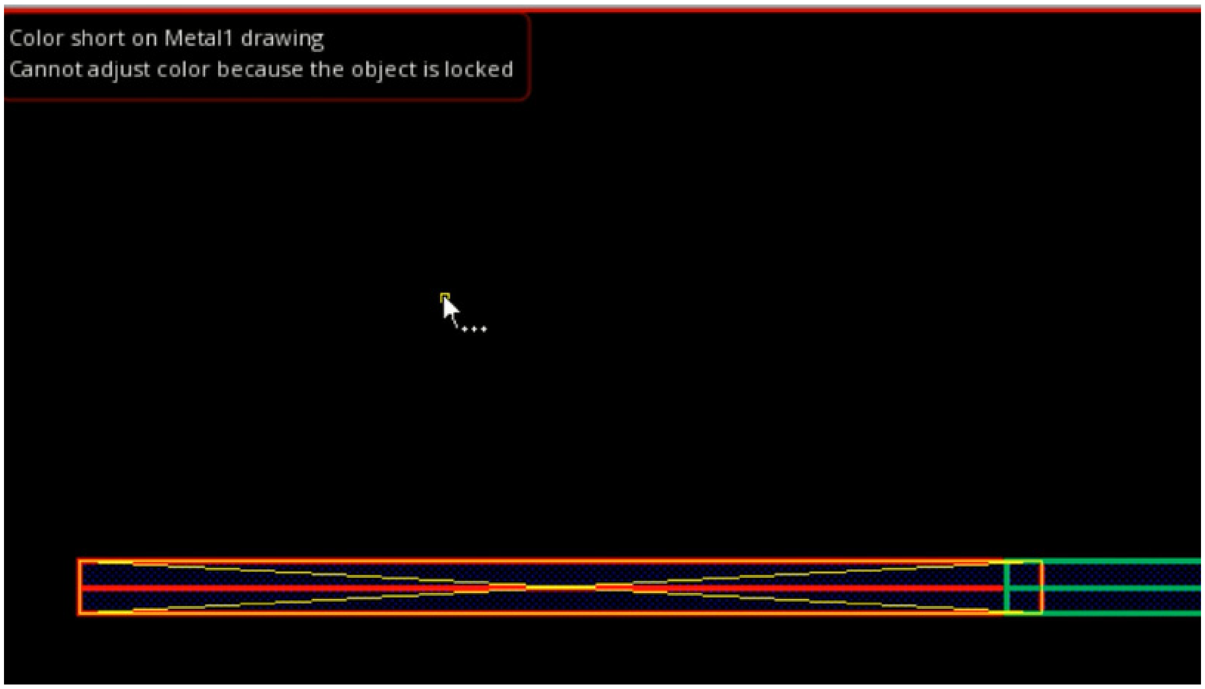
This section describes the options to view the last operation status and probing the color source.

Viewing the Last Operation Status

The color of the last operation status icon on the Multiple Patterning toolbar depicts the status of the last coloring operation. The colors depict the following:

- Blue: Indicates the last coloring operation was successful
- Red: Indicates the last coloring operation failed
- Yellow: Indicates that some operations were interrupted

When you click the red status button, temporary markers are generated on the violating shapes. You can click the marker on the shape to get more information about the coloring problem, as shown in the figure below.



Color short on Metal1 drawing
Cannot adjust color because the object is locked

Once you have analyzed the problem, you can press the `Esc` key to return to the normal mode and clear the temporary markers.

Probing Color Source

You can probe the color source and mark the color information on hierarchical color locks, clusters or layer shifts.

Probe Color Source

The *Probe Color Source* option displays the color information of the probed object. To display the color information for an object, choose *Probe Color Source* from the *Last Operation* drop-down list of the Multiple Patterning toolbar.

The following information is displayed:

- Basic Shape Information
- Effective Color and Lock State

- Cluster or Layer Shift
- Original Shape

The original shape information displays `uncolored`, `colored`, `trackColored`, `netclassPrecolored`, `userLocked`, and `systemLocked`. The `userLocked` or `systemLocked` are displayed at the level of the HCL. For locked via layers, this information is displayed at the level of the original shape.

When editing in place, the color information displayed is at the level of edit in place. An EIP suffix is appended at the level that is closest to the edit in place level. When the drawSurroundingOn environment variable is set to `t`, the color information is displayed at the top-level with the `EIP` suffix appended at the edit in place level. If the drawSurroundingOn environment variable is set to `nil`, the color information is displayed as if the probe is at the edit in place level. Also, no `EIP` suffix is appended.

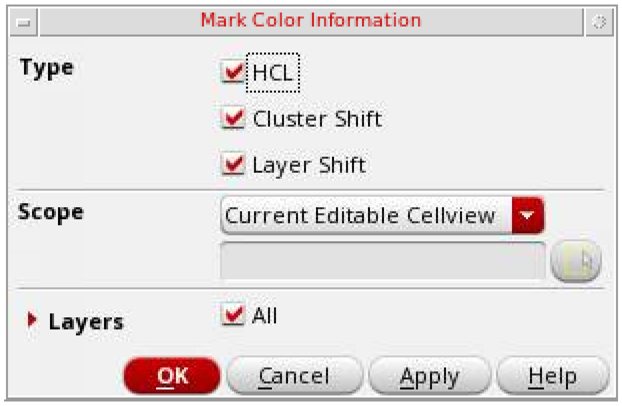
Mark Color Info

You can use the *Mark Color Info* option to place markers where hierarchical color locks have been dropped and on instances with color shifts. The color shifts could be cluster or layer shifts.

To mark the color information:

1. Choose *Mark Color Info* from the *Last Operation* drop-down list of the Multiple Patterning toolbar.

   The Mark Color Information form appears.



Mark Color Information

Type	✓ HCL
	✓ Cluster Shift
	✓ Layer Shift
Scope	Current Editable Cellview
Layers	✓ All

2. Select the *Type: HCL*, *Cluster Shift*, or *Layer Shift*.
3. Choose the *Scope: Current Editable Cellview*, *Current Visible Area*, or *Area*.

4. Select the *Layers: All* or *Selected Layers*.
5. Click *OK*.

The Mark Color Info Summary is displayed in the CIW.

Mark Variant Cells

You can use the *Mark Variant Cells* option to place markers on instances that will generate potential variants after XStream Out. It depends on the `layerMap` file and if unlocked, colors are streamed out.

To mark variant cells, choose *Mark Variant Cells* from the *Last Operation* drop-down list of the Multiple Patterning toolbar.

The variant cells summary is displayed in the CIW.

Note: The via variants will be ignored in the variant cells summary report.

Mark doNotColor Cells

You can use the *Mark doNotColor Cells* option to place markers on instances with cells that have the `mptDontColor` property. These cells are skipped by the coloring engine during recoloring. The `mptDontColor` property only affects the recoloring of the master. It does ot prevent color shifting at the instance level. For example, there is an instance `I1` of `Cell1` in the top design. If you set the `mptDontColor` property on `I1` or `Cell1`, and recolor the entire hierarchy, `Cell1` master is not recolored. However, color shifting of instance `I1` is not prevented.

Using the Multiple Patterning Color Engine

When the multiple patterning color engine is enabled, the coloring method specifies advanced methodologies that will be used to color shapes. Two coloring methods are available: *interactive* and *managed*. Features supported by these methods are described in the following sections:

- Hierarchical Coloring of Connected Shapes
- Automatic Color Shifting

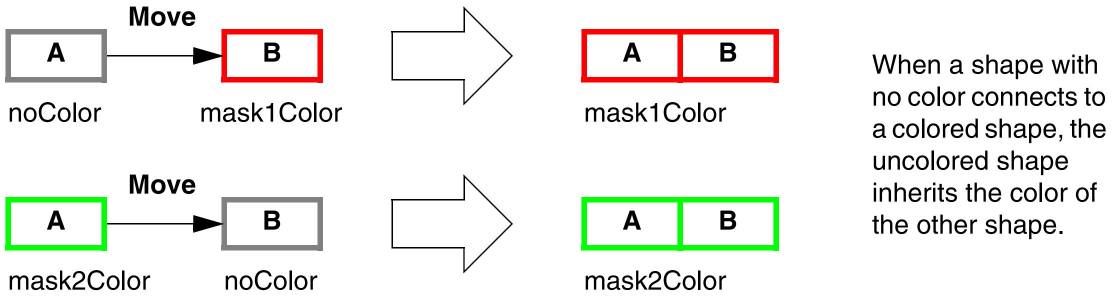
To use the multiple pattering color engine, follow the procedure in Enabling the Multiple Patterning Color Engine for turning on the color engine and setting the coloring method.

Video

For an overview of this feature, see Dynamic Coloring Utilities (Interactive and Managed).

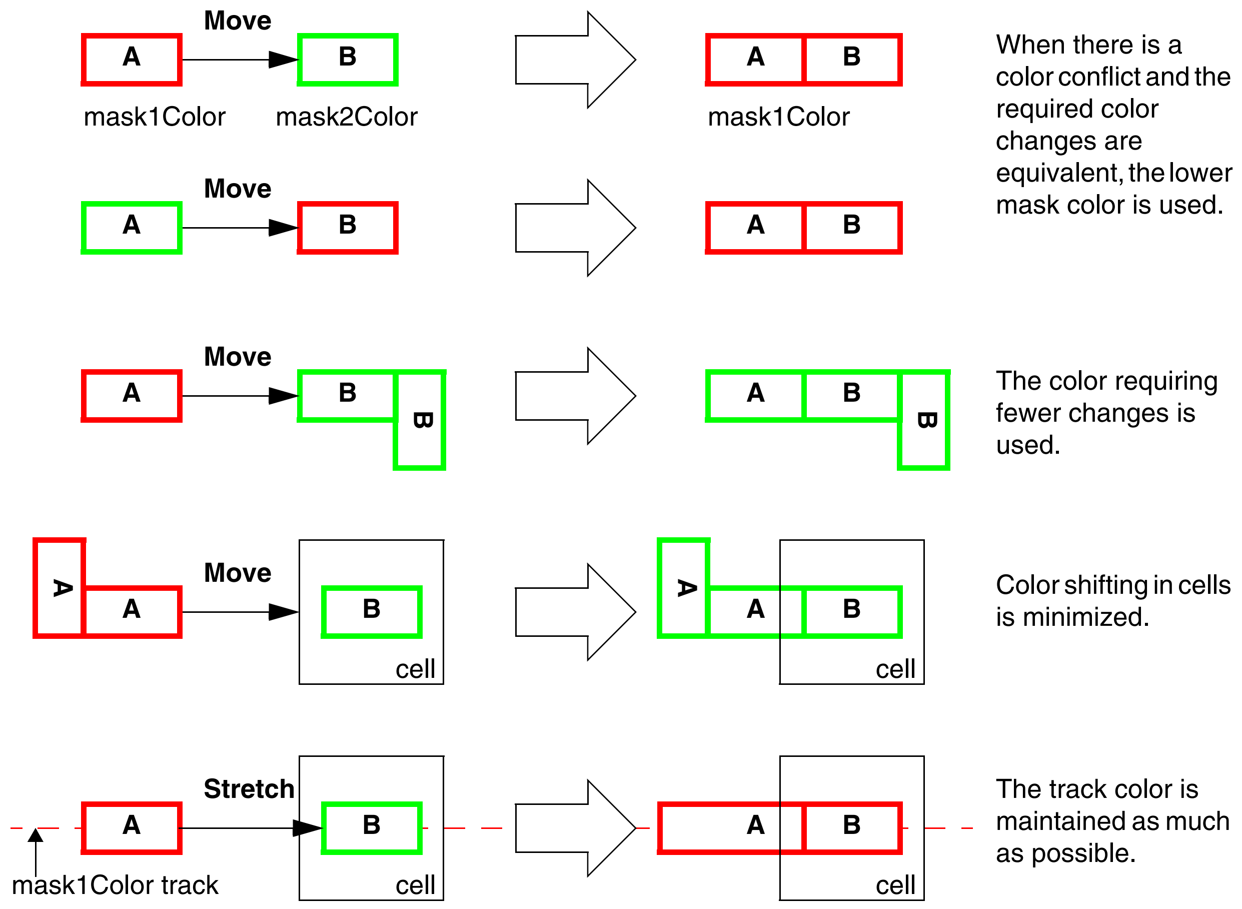
Hierarchical Coloring of Connected Shapes

When the multiple patterning color engine is enabled, connected shapes form a cluster that is managed by the color engine. Color is automatically propagated on the layer through the hierarchy when shapes are added to the cluster, either by creating new shapes or by moving or stretching existing shapes.


Move
A
B
noColor
mask1Color
A
B
mask1Color
When a shape with no color connects to a colored shape, the uncolored shape inherits the color of the other shape.
Move
A
B
mask2Color
noColor
A
B
mask2Color

When color is inherited from another shape, the color can be reversed using Undo. However, movement in the reverse direction will not automatically undo the color inheritance.

Color assignment is cost-based with priority given to fewer color changes when there is a conflict.

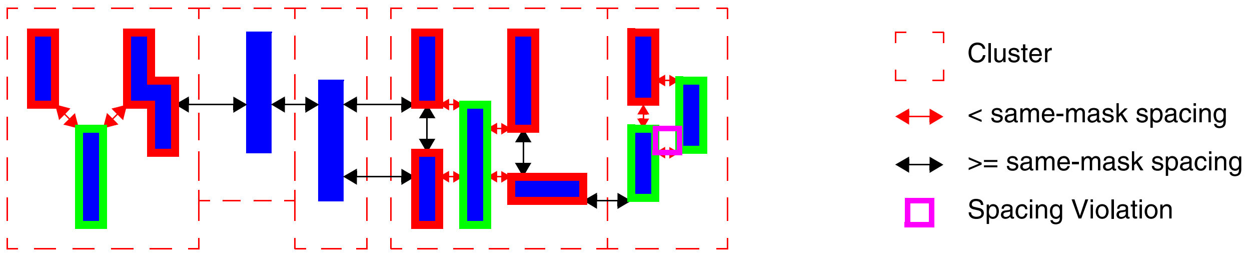

Move
A
B
mask1Color
mask2Color
A
B
mask1Color
When there is a color conflict and the required color changes are equivalent, the lower mask color is used.
Move
A
B
A
B
Move
A
B
B
A
B
B
The color requiring fewer changes is used.
Move
A
A
B
cell
A
A
B
cell
Color shifting in cells is minimized.
Stretch
A
B
cell
mask1Color track
A
B
cell
The track color is maintained as much as possible.

Automatic Color Shifting

When the color engine is enabled and the coloring method is **managed**, coloring is managed actively. As shapes and instances are added, removed, changed, or moved, those objects and neighboring objects will be colored according to the clustering algorithm. Automatic color shifting is performed on shapes, instances, and vias when objects are within same-mask spacing apart. Shapes that are greater than same-mask spacing from any another shape are not color shifted.

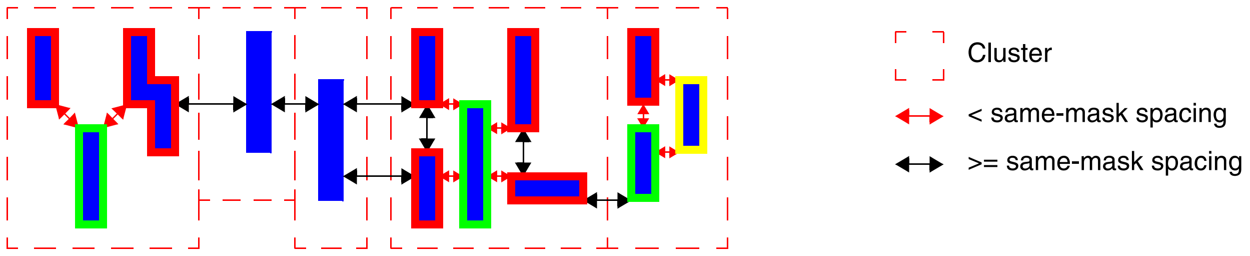
You cannot shift colors in a managed mode cluster if any of the shapes in the cluster is locked, except when *Allow Shifting of Locked Shapes* is enabled. For more information on setting this option, see Using the Multiple Patterning Options Form.

Figure 4-3 Double Patterning Cluster Coloring Example



For layers supporting three masks, the color engine will assign an unresolved color shape to mask3Color.

Figure 4-4 Multi-Patterning Cluster Coloring Example



Color Shifting for Cells

You can add the `colorShiftingPolicy` property to a cell master to override the shift types that the coloring engine can use to fix coloring conflicts.

```
colorShiftingPolicy "string" "cluster | layer | none"
```

- `cluster`: Cluster references are created.
- `layer`: Only layer shifts are used on the instances of the cell. Cluster references are not created.
- `none`: Instances of the cell are not shifted. Cluster references are not created.

When the `colorShiftingPolicy` property is not set, the globalColorShiftingPolicy environment variable is used to control the color shift type.

To create the property on a cell master, use the following command:

```
dbCreateProp(cv "colorShiftingPolicy" "string" "none" )
```

The settings are set at the session or cell level. The property overrides the global settings when the property is set.

Some examples of usage of this property are listed below:

```
globalColorShiftingPolicy = "none"
```

CELL1 has the `colorShiftingPolicy` property set to `layer`.

In the above example, instances of CELL1 are layer shifted.

```
globalColorShiftingPolicy = "layer"
```

CELL1 has the `colorShiftingPolicy` property set to `none`.

In the above example, instances of CELL1 cannot be shifted.

Track-Based Coloring

To use track-based coloring, you must create track patterns that assign mask colors in the layout. Shapes will be colored based on their position relative to the tracks. The Virtuoso wire editor can snap wires to tracks.

There are two methods for creating colored tracks in the layout:

- Track Patterns

  Track patterns specify the routing layer, direction, and offset. The spacing between tracks is fixed. The color pattern can be customized.

- Width Spacing Patterns (WSP)

  WSPs create a *correct by construction* non-uniform routing grid with pre-defined width and color constraints. This methodology supports the creation of a global grid and one or more regions, where the grid for each region can be different from the global grid for increased flexibility and productivity with complex designs. For information on creating grids using WSPs, see Using Width Spacing Patterns in *Virtuoso Width Spacing Patterns User Guide*.

To route wires on tracks with more than one mask per layer:

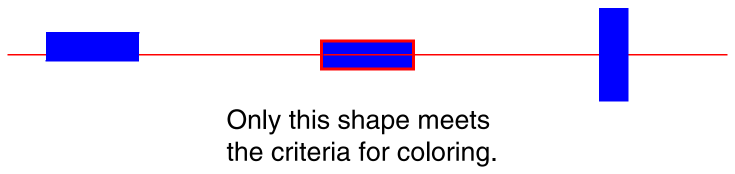
1. Create colored track patterns using one of these methods:
   - ❑ Creating Colored Tracks Using Track Patterns
   - ❑ Creating Colored Tracks Using Width Spacing Patterns
2. Turn on the multiple patterning color engine, as described in Turning the Multiple Patterning Color Engine On and Off.
3. Create wires snapped to tracks, as described in Creating Wires Snapped to Track Patterns in *Virtuoso Interactive and Assisted Routing User Guide*.

   The guidelines for coloring shapes are described in Coloring Shapes that Overlap Colored Tracks.

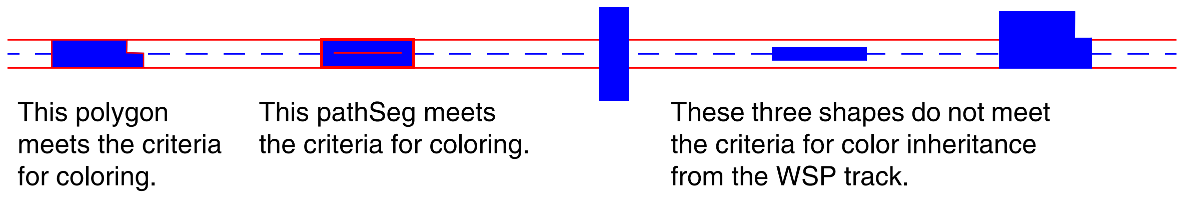
Coloring Shapes that Overlap Colored Tracks

By default, shapes inherit the color of a track depending on the track type and position of the shape relative to the track:

- For track patterns, the color of the track is inherited only if the centerline of the shape is aligned with the centerline of the track, as shown below.



- For WSP tracks, the color of the track is inherited only if the centerline of the wire or pathSeg and its edges align with the track's centerline and edges. For polygons and rectangles, color inheritance requires that the bounding box of the shape aligns with the track's edges and that the minimal length for the bounding box in the x or y direction equals the width of the track.


This polygon
meets the criteria
for coloring.
This pathSeg meets
the criteria for coloring.
These three shapes do not meet
the criteria for color inheritance
from the WSP track.

To change the default behavior so that all the shapes overlapping a colored track inherit the track's color, regardless of the alignment, set the trackColoringOnlySnappedShapes environment variable to `nil`.

```
envSetVal("mpt" "trackColoringOnlySnappedShapes" 'boolean nil)
```

For WSP tracks, you can restrict the coloring inheritance to *active* colored WSP tracks by setting the onlyCheckActiveWSP environment variable to `t`.

```
envSetVal("mpt" "onlyCheckActiveWSP" 'boolean t)
```

Creating Colored Tracks Using Width Spacing Patterns

To set up colored tracks using width spacing patterns (WSPs):

1. Follow the instructions in Preparing to Use Width Spacing Patterns to set the pattern-related constructs, purposes, and display packets in the technology file, and set the display resource file.

2. Use the Track Pattern Assistant to choose the WSPs for the global area and regions in the canvas. For information, see Working with the Track Pattern Assistant.

Creating Colored Tracks Using Track Patterns

For each track pattern, you must specify the routing layer, track pattern direction, starting position with respect to the origin, spacing between tracks, and the total number of tracks. You can optionally assign a name to the track pattern, and customize the color pattern.

Color patterns can be specified using one of the following:

- A mask color (`R` for mask1Color and `G` for mask2Color; uppercase and lowercase characters are accepted)
- An alternating color pattern (for example, `"RG"` or `"GR"`)
- A sequence of mask colors (for example, `"RGR"`)
- Repeating patterns given in compact notation "(*`pattern`*)^*`count`*"

  where *`pattern`* is a sequence of mask colors and *`count`* is an integer representing the number of times the sequence is repeated. For example, `"(RG)^3(R)^8"` is equivalent to `"RGRGRGRRRRRRRR"`.

To control the display of track patterns in the layout view, refer to Displaying Track Patterns in Layout.

To create colored track patterns:

➡ Choose *Create – P&R Objects – Track Patterns* in the layout window.

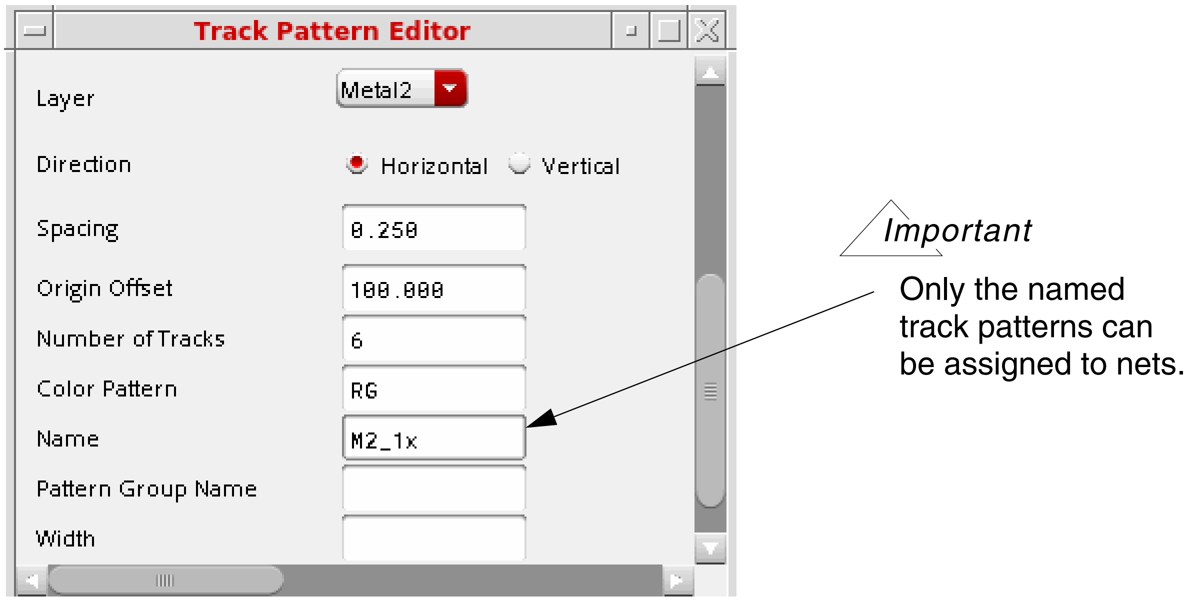
Refer to Track Pattern Editor Form for information on using this form.

The following examples illustrate how track patterns are created:

- Defining a Track Pattern with Alternating Colors
- Defining a Track Pattern with a Color Pattern
- Defining Multiple Track Patterns for a Single Layer

Example 4-1 Defining a Track Pattern with Alternating Colors

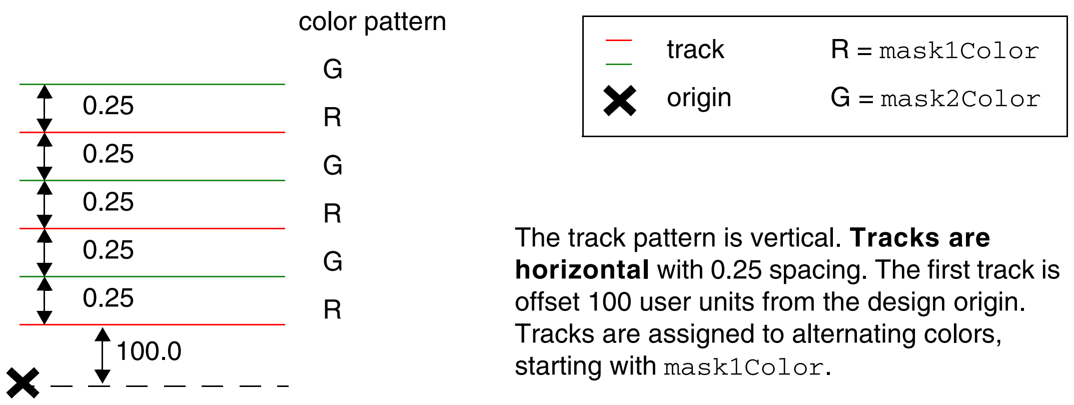
This example creates horizontal tracks on the `Metal2` layer, spaced 0.25 user units apart, with the first track offset 100 user units from the origin, as shown in Figure 4-5. The first track is `mask1Color`, the second is `mask2Color`, and so on, alternating colors ("RG") until 6 tracks have been created.



Layer	Metal2
Direction	[✓ Horizontal] Vertical
Spacing	0.250
Origin Offset	100.000
Number of Tracks	6
Color Pattern	RG
Name	M2_1x
Pattern Group Name	
Width	

Only the named track patterns can be assigned to nets.

Figure 4-5 Illustration of a Track Pattern with Alternating Colors

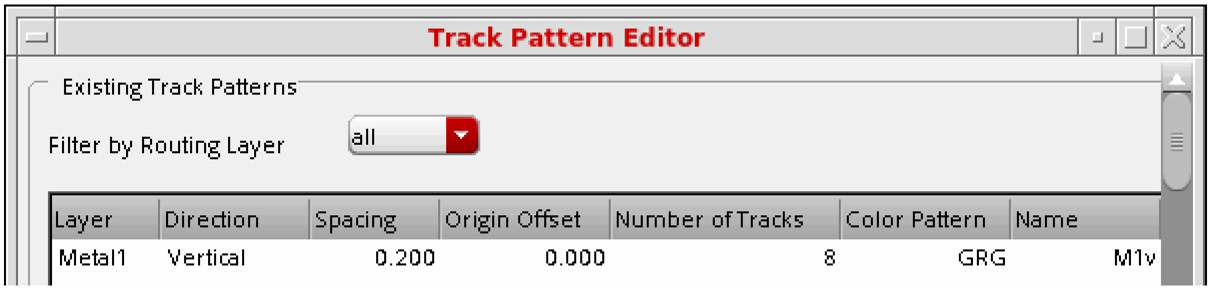


The track pattern is vertical. **Tracks are horizontal** with 0.25 spacing. The first track is offset 100 user units from the design origin. Tracks are assigned to alternating colors, starting with `mask1Color`.

Example 4-2 Defining a Track Pattern with a Color Pattern

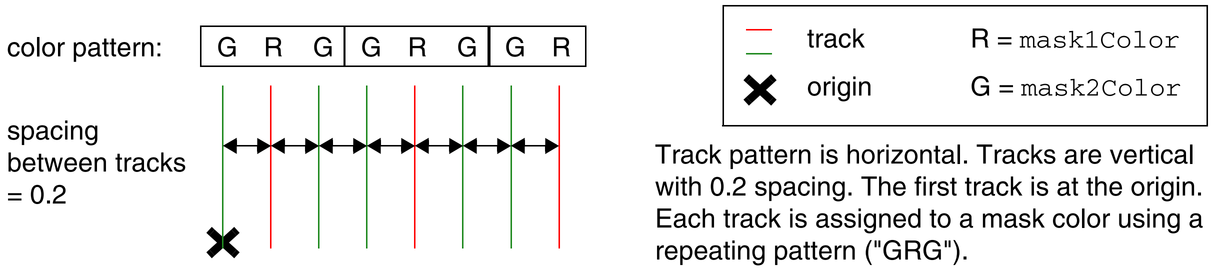
This example creates vertical tracks on the `Metal1` layer, spaced 0.2 user units apart, with the first track at the origin. The tracks are assigned to color masks in this sequence:

`mask2Color`, `mask1Color`, `mask2Color`. The color track pattern is repeated until 8 tracks have been created, as shown in Figure 4-6.



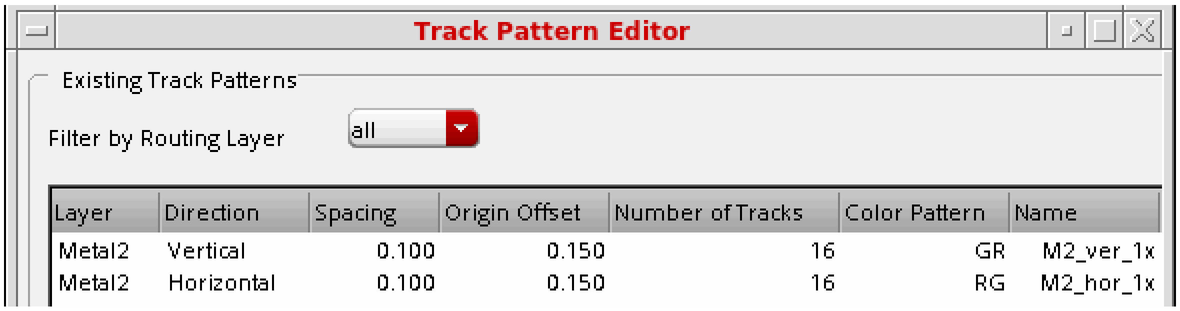
Layer	Direction	Spacing	Origin Offset	Number of Tracks	Color Pattern	Name
Metal1	Vertical	0.200	0.000	8	GRG	M1v

Figure 4-6 Illustration of a Track Pattern with a Color Pattern


color pattern:
G R G G R G G R
spacing between tracks = 0.2
track
origin
R = mask1Color
G = mask2Color
Track pattern is horizontal. Tracks are vertical with 0.2 spacing. The first track is at the origin. Each track is assigned to a mask color using a repeating pattern ("GRG").

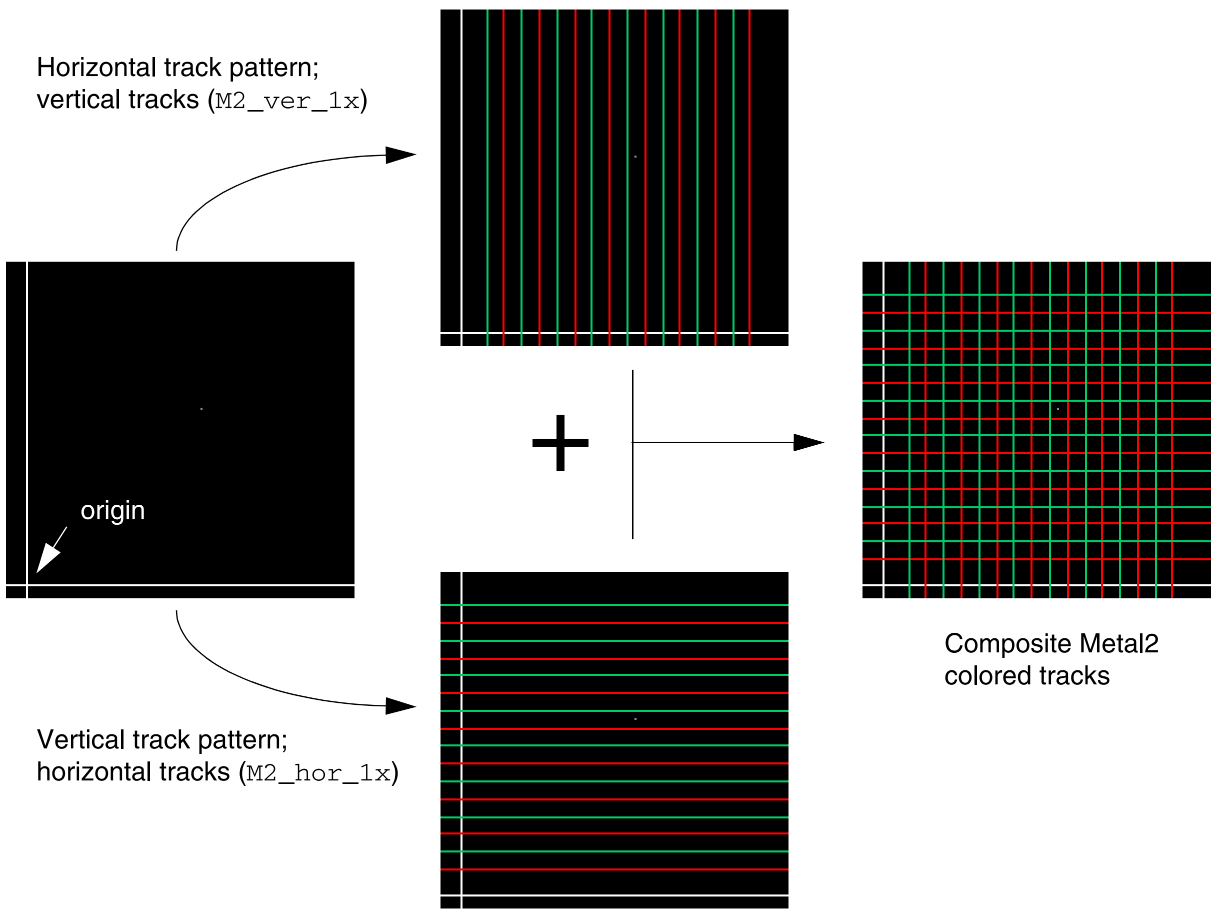
Example 4-3 Defining Multiple Track Patterns for a Single Layer

This example creates horizontal and vertical tracks on the `Metal2` layer, spaced 0.1 user units apart with the first track offset 0.15 user units from the origin in both directions. The tracks are assigned to alternating color masks, starting with `mask1Color` for the horizontal tracks, and `mask2Color` for the vertical tracks. Sixteen (16) tracks are created in each direction, as shown in Figure 4-7.



Layer	Direction	Spacing	Origin Offset	Number of Tracks	Color Pattern	Name
Metal2	Vertical	0.100	0.150	16	GR	M2_ver_1x
Metal2	Horizontal	0.100	0.150	16	RG	M2_hor_1x

Figure 4-7 Illustration of Multiple Track Patterns for a Single Layer


Horizontal track pattern;
vertical tracks (M2_ver_1x)
origin
Vertical track pattern;
horizontal tracks (M2_hor_1x)
Composite Metal2
colored tracks

Displaying Track Patterns in Layout

To show or hide all the track patterns or track patterns by layer, follow the procedure in Displaying Track Patterns in *Virtuoso Layout Suite L User Guide*. All track patterns that are set visible in the Palette assistant are drawn in the active layout cellview.

Assigning Track Patterns to Nets

To route a net on specific tracks, follow the procedures in these sections:

- Assigning Track Patterns to Constraint Groups
- Applying Constraint Groups to Nets

Assigning Track Patterns to Constraint Groups

After creating the track patterns, assign them as constraints to a constraint group for track-based routing, in the same way you use spacing and width constraints to define spacing and width characteristics.

Important

Only the named track patterns can be assigned to a constraint group.

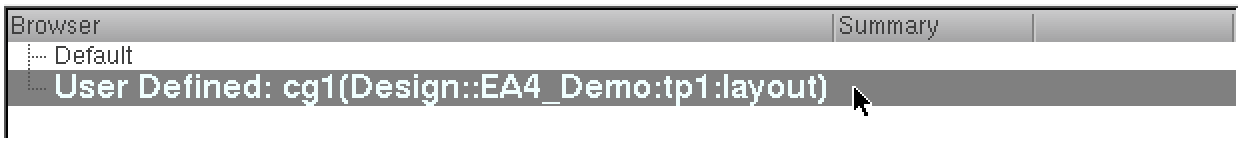
To assign track patterns to constraint groups:

➡ Use the Process Rule Editor from the Constraint Manager to add track pattern constraints to a constraint group.

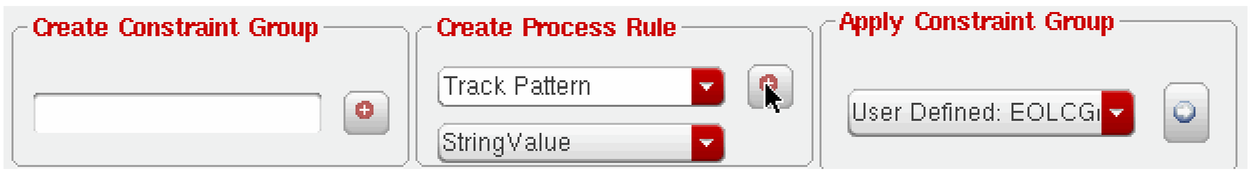
Refer to *Virtuoso Unified Custom Constraints User Guide* for information on accessing the Constraint Manager, starting the Process Rule Editor, creating a constraint group, and creating a new process rule.

Example 4-4 Assigning track patterns to a constraint group

1. Follow the procedure "*Starting the Process Rule Editor*" in *Virtuoso Unified Custom Constraints User Guide* to start the Process Rule Editor.
2. (Optional) Create a new constraint group.
3. In the Browser, click the name of the constraint group.

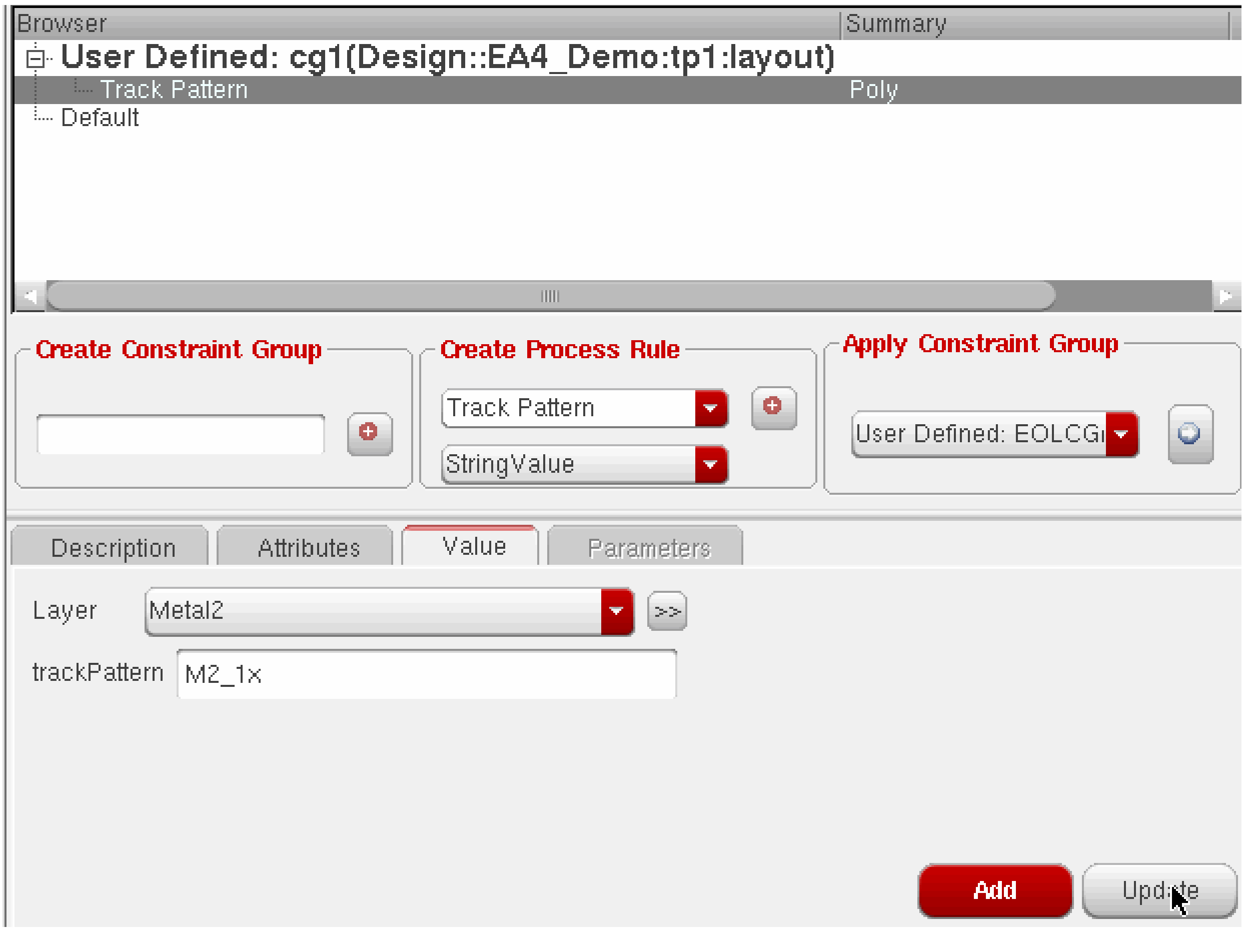


4. In the *Create Process Rule* field, choose *Track Pattern* from the drop-down list, then click the plus (+) button.



5. With the track pattern selected in the Browser, click the Value tab at the bottom of the form.
6. Choose the *Layer* for the tracks from the drop-down list.

7. In the *trackPattern* field, specify the name of the track pattern.



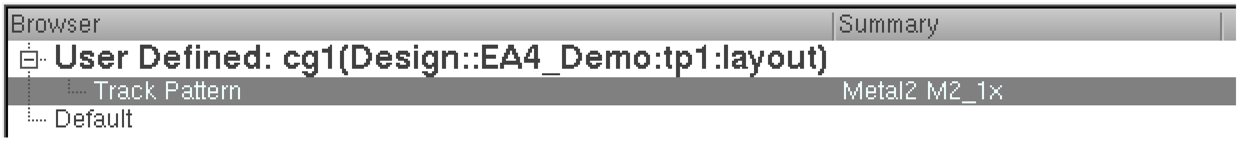
Layer: Metal2
trackPattern: M2_1x

Important

If more than one track pattern is needed for the layer (for example, when the layer has bidirectional tracks), all the track patterns must be specified in a single track pattern constraint. To do this, separate the track pattern names using a plus (+) with no added spaces (for example, `M1_hor_1x+M1_ver_1x`).

8. Click *Update.*

The Browser is updated with the layer name and track pattern.



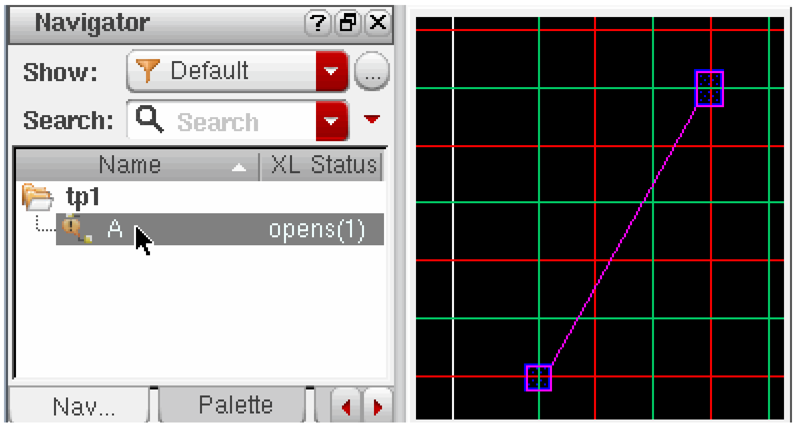
Browser	Summary
User Defined: cg1(Design::EA4_Demo:tp1:layout)	
Track Pattern	Metal2 M2_1x
Default	

Applying Constraint Groups to Nets

After adding track patterns to a constraint group, you can now apply the constraint group to a net. The net will be routed on the specified tracks if you enable snapping to track patterns.

Example 4-5 Applying a constraint group to a net

1. In the Navigator assistant, select the net.



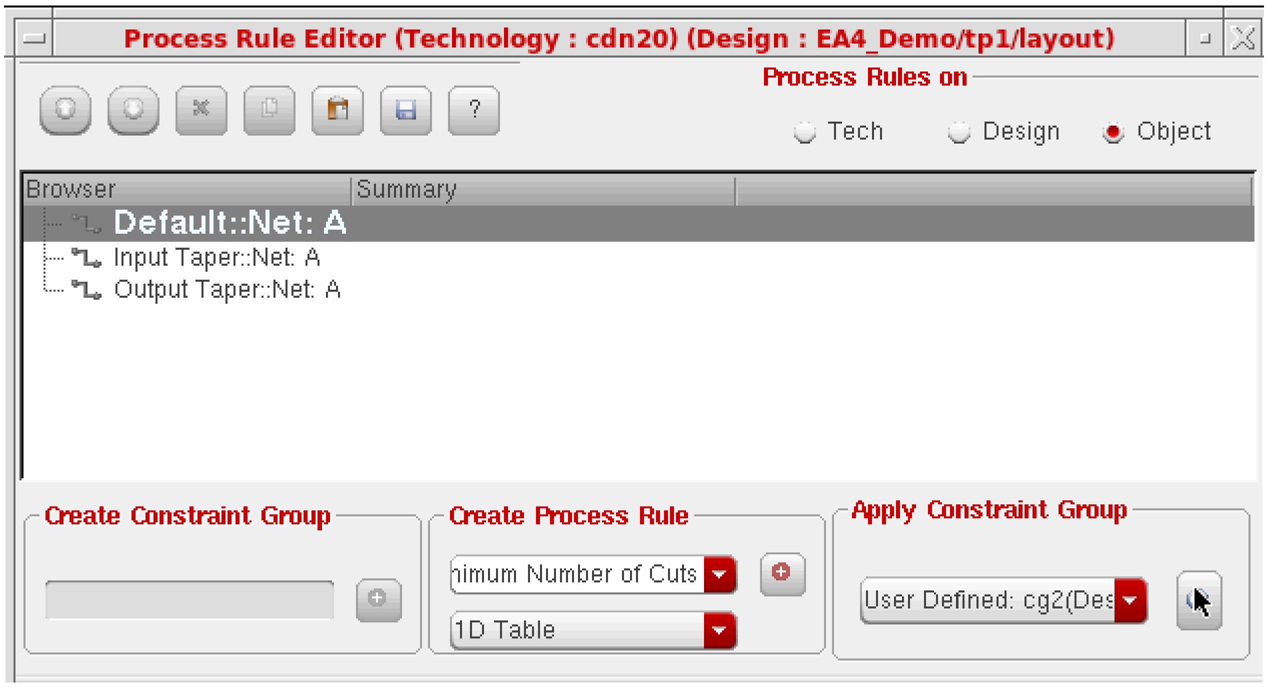
Navigator

Show: Default

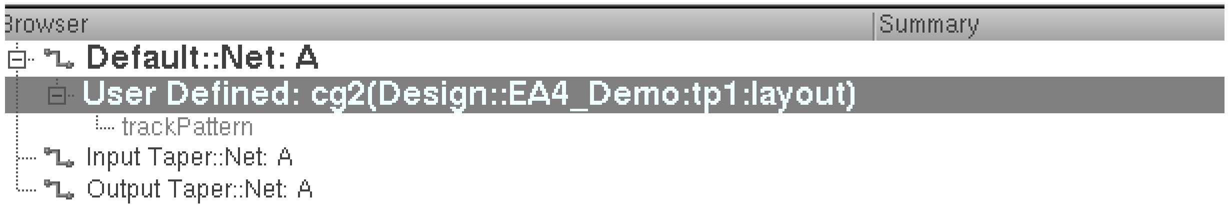
Search: Search

Name	XL Status
tp1	
A	opens(1)

2. In the Process Rule Editor *Apply Constraint Group* section, choose the constraint group with the track pattern from the drop-down list, then click the *Apply* button.



The constraint group appears in the browser. Expand the constraint group to view the included constraints.



The net can now be routed on the specified tracks using the Virtuoso wire editor, as described in Creating Wires Snapped to Track Patterns in *Virtuoso Interactive and Assisted Routing User Guide*.

Constraint Group Support to Define Colored LPPs

The coloring engine uses the constraint group with `virtuosoMPTSetup` as a constraint group definition in the technology file to determine the colored layer-purpose pairs. When a valid constraint group is found in the technology file, the colored layer-purpose pairs are defined according to the definition in the constraint group.

You can use the environment variable, mptConstraintGroup, to specify the constraint group to be used to determine colored LPPs.

The `validLayers` constraint lets you restrict the colored layers. The example below shows the use of the `validLayers` constraint to limit the colored layers to one or more specific layer-purpose pairs.

```
constraintGroups(
  ("virtuosoDefaultMPTSetup" nil "virtuosoMPTSetup"
   interconnect(
     ( validLayers   (M1 V1 (M2 pin)))
   )
```

In the above example:

- M1: All purposes are colorable
- V1: All purposes are colorable
- M2: Only `pin` purpose is colorable

The `validPurposes` constraint lets you define the colored purposes by including or excluding a set of purposes for a layer. The example below shows the use of the `validPurposes` constraint.

```
constraintGroups(
  ("virtuosoDefaultMPTSetup" nil "virtuosoMPTSetup"
   interconnect(
     ( validLayers    (M1 (M2 drawing)))
      (validPurposes  ('include (pin))
   )
  )
```

In the above example:

- M1: Only `pin` purpose is colorable
- M2: Only `drawing` purpose is colorable

Note: The `validPurposes` constraint only applies to layers specified in the `validLayers` constraint. It does not apply to layer-purpose pairs specified in the `validLayers` constraint. This is the reason in the above example, only the drawing `purpose` is colorable and `pin` purpose is not colorable.

The `excludeLPPs` constraint lets you exclude LPPs from the colorable layer-purpose pair list derived from the combination of `validLayers` and `validPurposes` constraints.

```
 constraintGroups(
   ("virtuosoDefaultMPTSetup" nil "virtuosoMPTSetup"
    interconnect(
      ( validLayers    (M1 (M2 ))
      (validPurposes  ('include (pin drawing))
      (excludeLPPs (M1 pin) (M2)))
   )
 )
```

In the above example:

- M1: Only `drawing` purpose is colorable
- M2: No purpose is colorable

In case the constraint group, `virtuosoMPTSetup`, is not found in the technology file, the layers with the number of masks more than one are considered as colorable layers.

The coloring engine uses the default colors specified in the `layerDefaultColor` constraint in the technology file defined in the MPT constraint group.

```
constraintGroups(
  ("virtuosoDefaultMPTSetup" nil "virtuosoMPTSetup"
    spacings(
    ( layerDefaultColor  "M1"  "mask2" )
   ( layerDefaultColor  "M2"  "mask1" )
   ) ;spacings
  )
)
```

In the above example:

- `mask2` is defined as default color for M1
- `mask1` is defined as default color for M2

Note: The default color set to gray is implies an undefined default color.

Migrating from the Multiple Patterning Assistant

Note: The Multiple Patterning assistant was removed in the base release of ICADV12.3.

You can access the Multiple Patterning assistant functions by using other Virtuoso tools and functions, as shown in the table below.

Multiple Patterning Assistant Functionality	**Alternative Method**	**For more information, refer to:**
Toolbar functions, showing and hiding color	Multiple Patterning toolbar	Using the Multiple Patterning Toolbar
Show the color and color state of the selected shapes	Selection Info toolbar, Property Editor, Dynamic Selection Assistant	Inspecting Colored Data
Showing and hiding shapes based on their color properties	Palette Assistant with *MPT Support* enabled	Controlling the Visibility and Selectability of Colored Data
Working with color groups	SKILL functions	Relationship Coloring

Saving and Restoring Data

Coloring information is saved with the data OpenAccess multi-patterning format and restored when the data is reloaded.

Limitations

Virtuoso MPT is specifically targeted for interactive drawing of a layout and operates best on only a few hundred shapes or objects. Scripts that create a large number of shapes will run slowly.

If pre-coloring is not used and/or colors are not locked for interactive/batch coloring, then the displayed colors do not represent mask assignments. For these cases, sign-off and mask making should be done by a batch mode geometry processing tool such as Cadence Physical Verification System. These tools can be used to address issues such as balancing the densities of the masks.

5

Checking and Fixing Multiple Patterning Violations in Layout

Virtuoso® Multi-Patterning Technology (MPT) in Layout Editor lets you:

- Check for multiple patterning violations using Virtuoso® Design Rule Driven (DRD) Edit.
- Perform in-design verification using Pegasus Interactive to improve design quality and increase productivity.
- Fix multiple patterning violations by moving, stretching, splitting, and stitching shapes.

The following topics are included in this chapter:

- Types of Color Checks
- Checking for Multiple Patterning Violations
  - Using Color-Aware DRD Edit
    - Checking for Multiple Patterning Violations Using DRD
  - Using Pegasus Interactive
- Using the Annotation Browser to View Multiple Patterning Violations
- Fixing Multiple Patterning Violations
- Verifying the Consistency of Color Assignments

Types of Color Checks

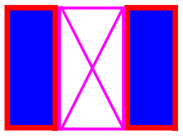
Color checks on a layout are categorized as follows:

- Color Rule Violations
- Color Shorts
- Uncolored Shapes
- Colorability

Color Rule Violations

Same-mask spacing violations occur when the distance between two shapes of the same color is less than the same-mask spacing. Different-mask (diff-mask) spacing violations occur when the distance between two shapes of different masks is less than the diff-mask spacing. Same-mask spacing is less than diff-mask spacing.

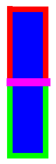
For more information on the constraints used to specify same-mask and diff-mask spacing, refer to Specifying Same-Mask and Diff-Mask Spacing Constraints.



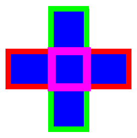
Same-mask spacing violation
Two shapes of the same color are less than same-mask spacing apart.

Color Shorts

Color shorts are overlaps between shapes with different colors, irrespective of their lock status.



Color Short Abutment
Two abutting shapes of different colors



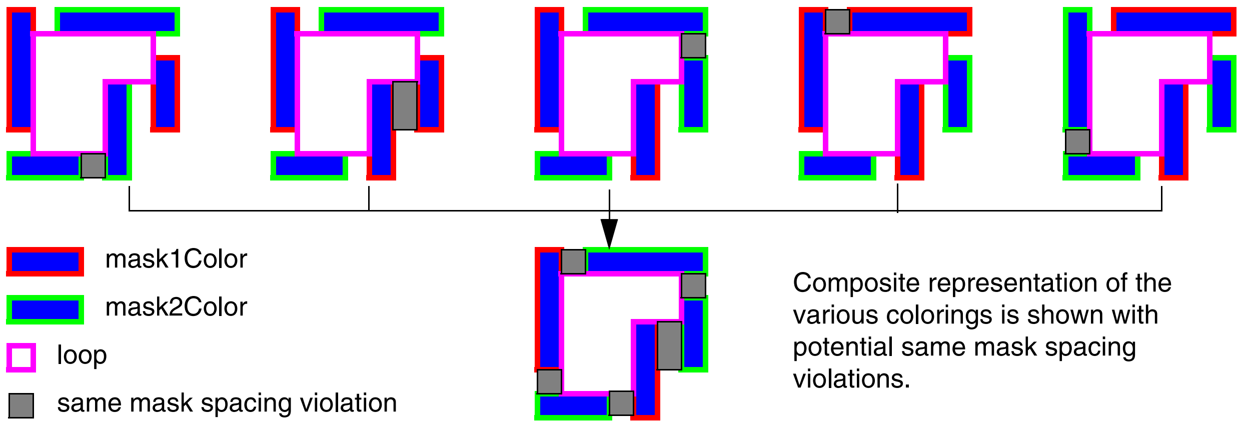
Color Short Overlap
Two overlapping shapes of different colors

Uncolored Shapes

Uncolored shapes on a multiple mask layer can be reported.

Colorability

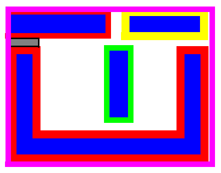
Colorability violations occur when the layout topology results in two adjacent shapes in a set of shapes on the same layer to have the same color. This condition is called an *odd cycle loop* in double patterning technology and is caused by an odd number of shapes in the set. The following example shows an odd cycle loop with different colorings applied, but none of them satisfies the same-mask spacing requirement for all the shapes.



Composite representation of the various colorings is shown with potential same mask spacing violations.

A similar conflict can occur with triple patterning technology, as shown below.

🟥	mask1Color
🟩	mask2Color
🟨	mask3Color
☐	triple patterning conflict
■	same mask spacing violation



Checking for Multiple Patterning Violations

The following sections describe two ways to check for multiple patterning violations:

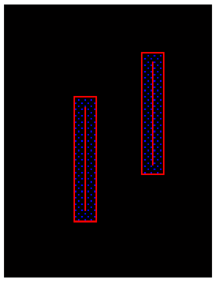
- Using Color-Aware DRD Edit
- Using Pegasus Interactive

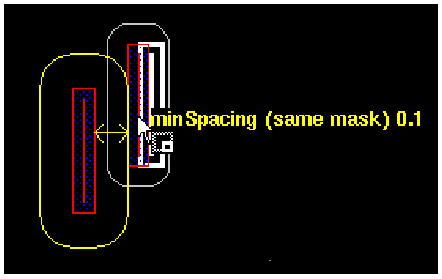
Using Color-Aware DRD Edit

Important

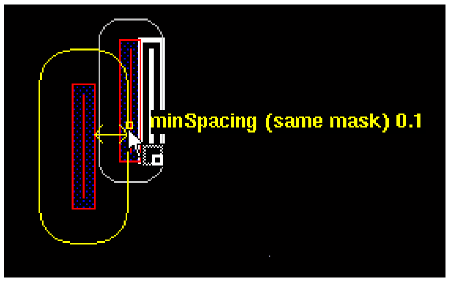
To use DRD to check for multiple patterning violations, ensure that same-mask spacing constraints are set in the technology database. For more information, see Specifying Same-Mask and Diff-Mask Spacing Constraints.

In Enforce and Notify modes, DRD can provide visual feedback to prevent certain types of multiple patterning violations.


Original same-layer and same mask shapes



Enforce mode is enabled. The right shape cannot be moved closer than same-mask spacing to the left shape.



Notify mode is enabled. The same-mask spacing halo is visible whenever the shapes are within same-mask spacing.

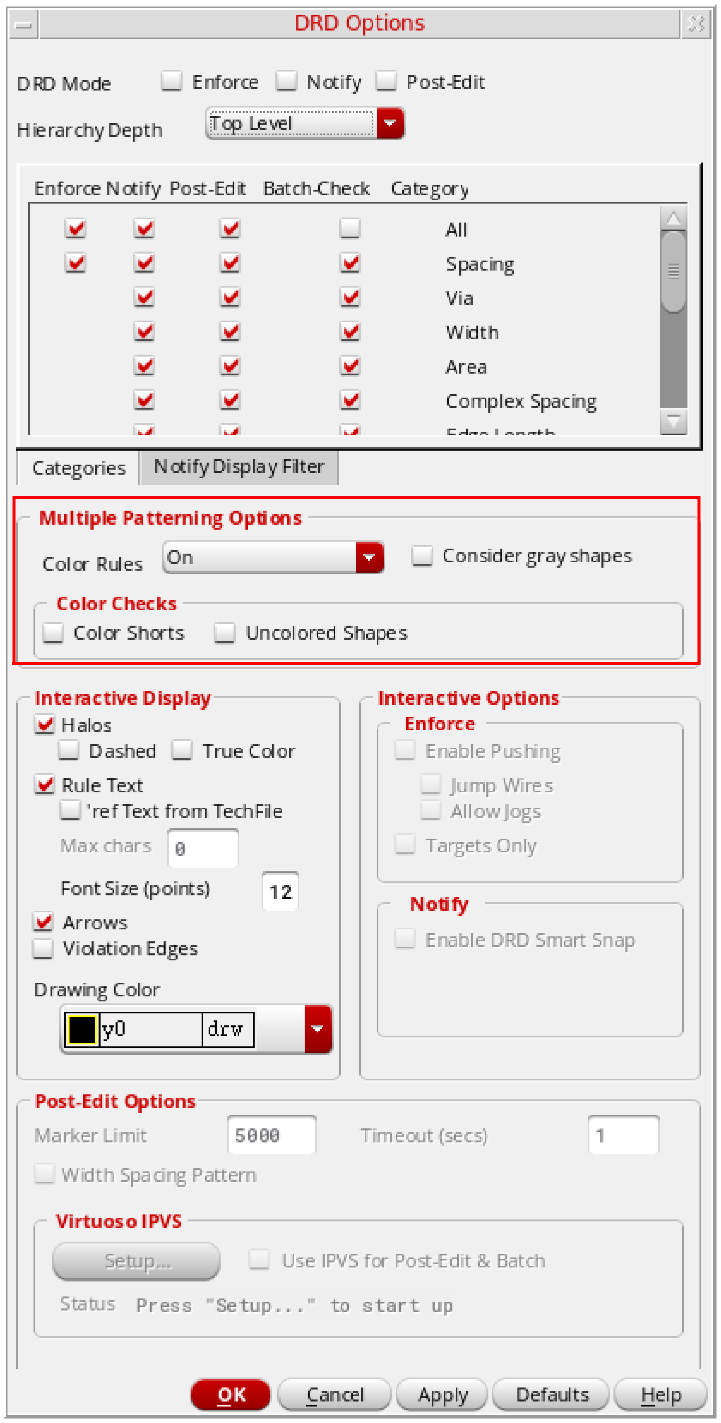
In Post-Edit and Batch modes, DRD can **report** multiple patterning violations as annotation markers in the workspace and in the Annotation Browser.

Checking for Multiple Patterning Violations Using DRD

To check for color violations in enforce, notify, or post-edit modes,

1. Choose *Options – DRD Edit.*

The DRD Options form appears.



DRD Mode ☐ Enforce ☐ Notify ☐ Post-Edit

Hierarchy Depth: Top Level

Enforce	Notify	Post-Edit	Batch-Check	Category
✓	✓	✓	☐	All
✓	✓	✓	✓	Spacing
	✓	✓	✓	Via
	✓	✓	✓	Width
	✓	✓	✓	Area
	✓	✓	✓	Complex Spacing

Multiple Patterning Options

Color Rules: On ☐ Consider gray shapes

Color Checks

☐ Color Shorts ☐ Uncolored Shapes

Interactive Display

✓ Halos
☐ Dashed ☐ True Color
✓ Rule Text
☐ 'ref Text from TechFile
Max chars: 0
Font Size (points): 12
✓ Arrows
☐ Violation Edges
Drawing Color: y0 drw

Interactive Options

Enforce

☐ Enable Pushing
☐ Jump Wires
☐ Allow Jogs
☐ Targets Only

Notify

☐ Enable DRD Smart Snap

Post-Edit Options

Marker Limit: 5000 Timeout (secs): 1
☐ Width Spacing Pattern

Virtuoso IPVS

☐ Use IPVS for Post-Edit & Batch
Status: Press "Setup..." to start up

2. Choose one or more DRD modes (*Enforce*, *Notify*, or *Post-Edit*).

3. Choose *Multiple Patterning Options*, as described in DRD Color Checks in *Virtuoso Design Driven Editing User Guide*.

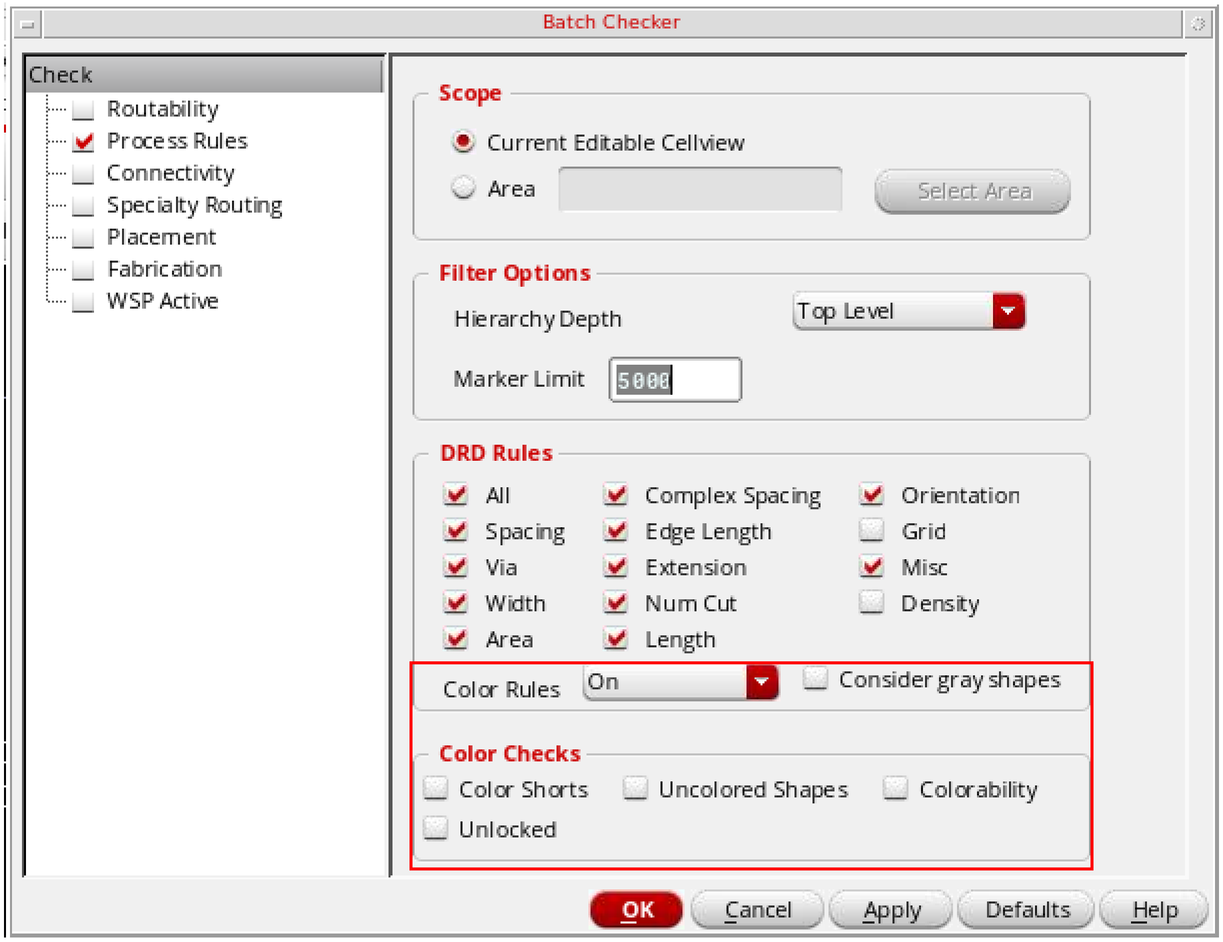
4. Click *OK* or *Apply*.

   As you modify the design, annotations will appear when multiple patterning violations are detected. If *Post-Edit* was selected, annotation markers can also be viewed in list form in the Annotation Browser. For more information, see Using the Annotation Browser to View Multiple Patterning Violations.

To check for color violations in batch mode,

1. Choose *Verify – Design*.

   The Batch Checker form appears.



2. Choose *Color Rules* and *Color Checks*, as described in DRD Color Checks in *Virtuoso Design Driven Editing User Guide*.

3. Click *OK* or *Apply*.

DRD checks the design in batch mode.

For information on how to view the results, see Using the Annotation Browser to View Multiple Patterning Violations.

Using Pegasus Interactive

Pegasus Interactive uses sign off DRC rules to verify the design. The rule deck must include multiple patterning rules in order to check for coloring violations.

Pegasus Interactive checking is run on demand from the Pegasus Interactive toolbar in Virtuoso. The violations, generated by Pegasus Interactive can be displayed as markers in the layout and the Annotation Browser, as described in Using the Annotation Browser to View Multiple Patterning Violations.

For details on using Pegasus Interactive, including requirements, setup, the Pegasus Interactive toolbar, and customizing the rule deck and the environment, see Getting Started in the *Pegasus Interactive User Guide*.

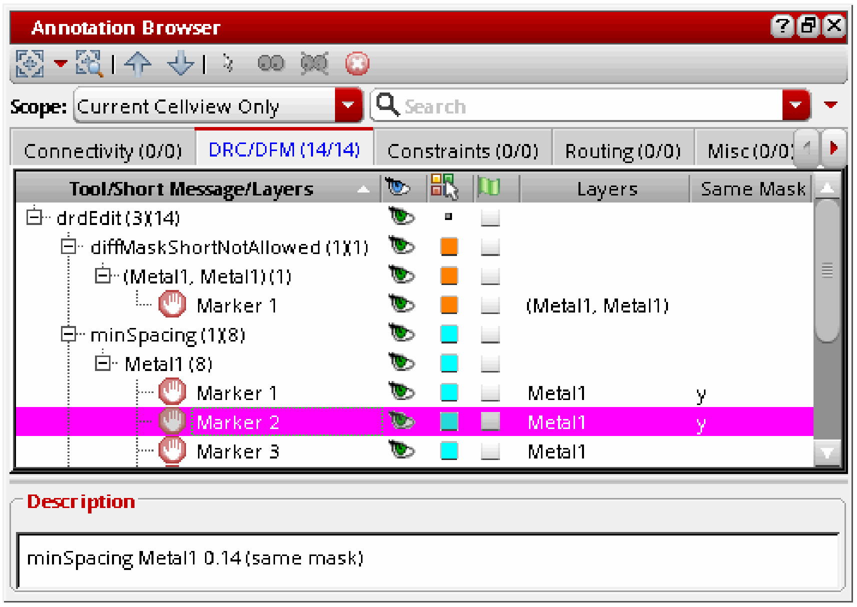
Using the Annotation Browser to View Multiple Patterning Violations

Use the Annotation Browser for more information on multiple patterning violations found by DRD or Pegasus Interactive.

To open the Annotation Browser,

1. Choose *Window – Assistants – Annotation Browser*.

   You can undock the Annotation Browser and locate it elsewhere on your desktop for better viewing.



Scope: Current Cellview Only

Tool/Short Message/Layers	Layers	Same Mask
drdEdit (3)(14)		
diffMaskShortNotAllowed (1)(1)		
(Metal1, Metal1) (1)		
Marker 1	(Metal1, Metal1)	
minSpacing (1)(8)		
Metal1 (8)		
Marker 1	Metal1	y
Marker 2	Metal1	y
Marker 3	Metal1	

minSpacing Metal1 0.14 (same mask)

2. Choose the DRC/DFM tab.

   Annotations are grouped by tool and type:

   ❑ *drdEdit*

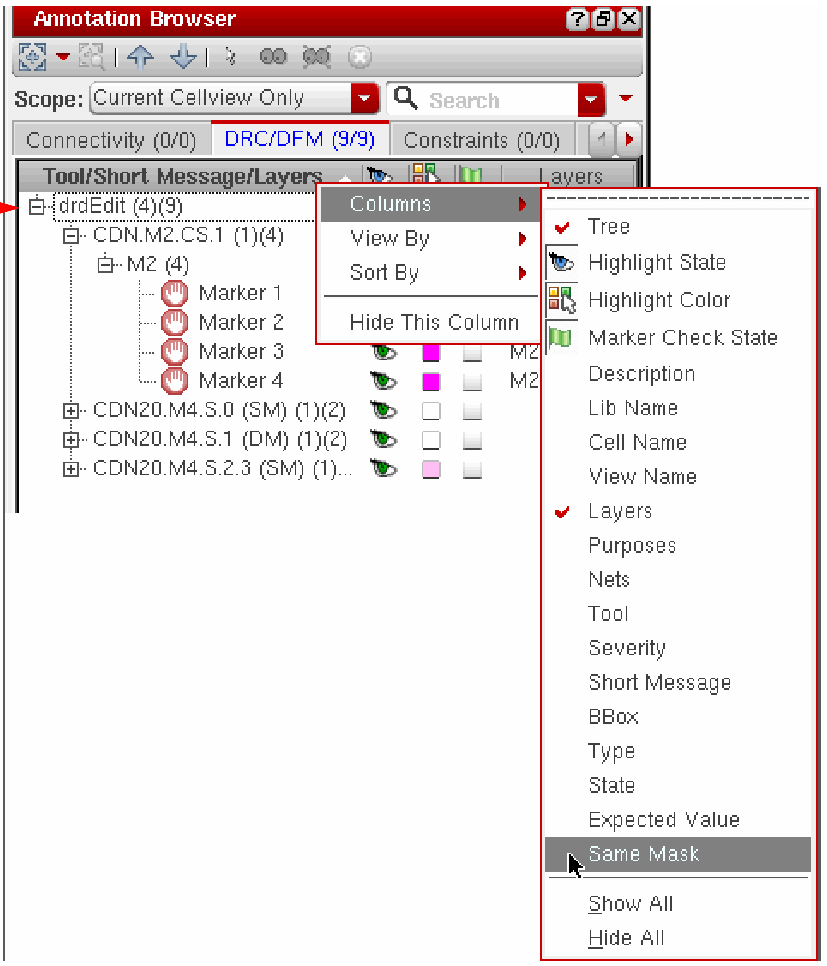
   These annotations are created by Virtuoso DRD.

   ❑ *Integrated Pegasus Interactive*

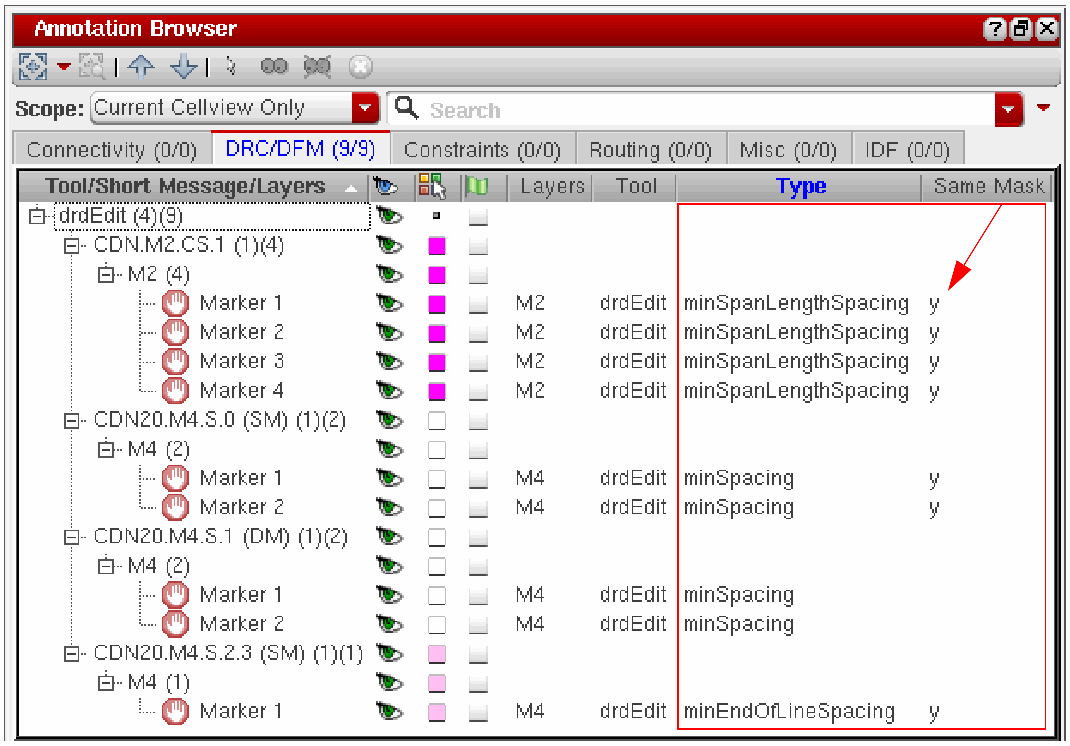
   These annotations are created by Pegasus Interactive.

3. (optional) Customize the Browser Pane as described in *Annotation Browser* in the *Virtuoso Layout Suite XL Reference Guide*.

Click to expand

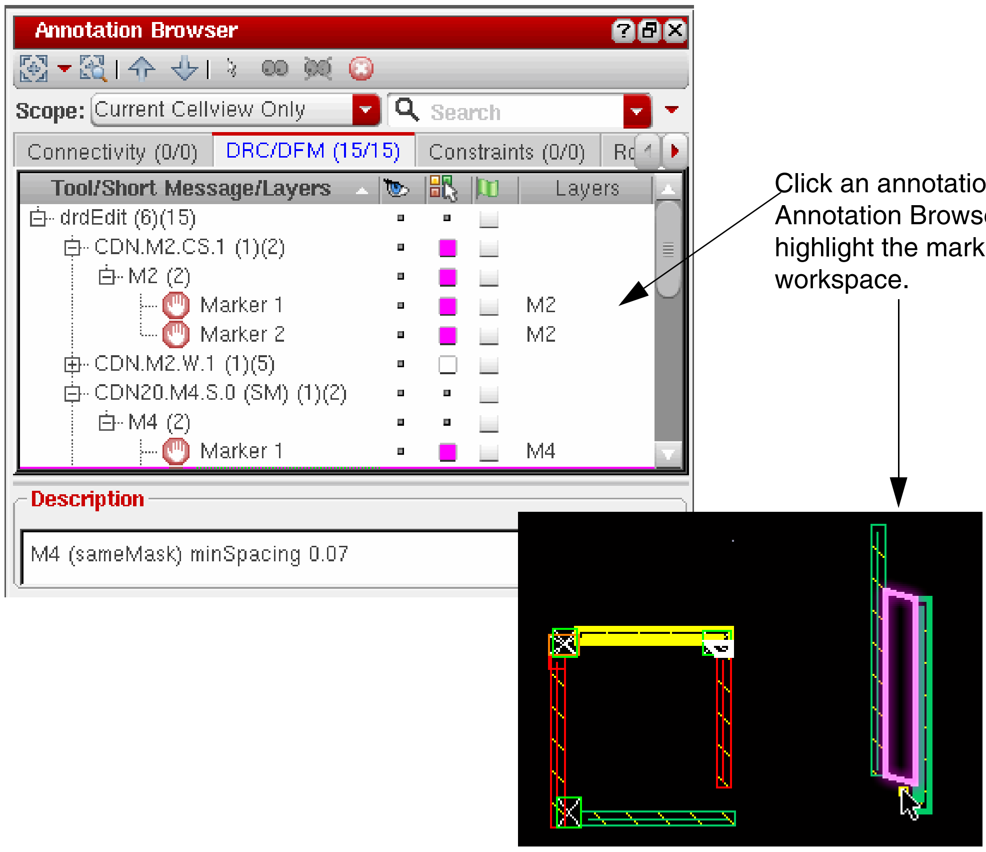


For *drdEdit* spacing violations, the *Same Mask* column will identify whether the constraint represents a same-mask spacing violation.



Tool/Short Message/Layers		Layers	Tool	Type	Same Mask
drdEdit (4)(9)	▪				
CDN.M2.CS.1 (1)(4)	🟪				
M2 (4)	🟪				
Marker 1	🟪	M2	drdEdit	minSpanLengthSpacing	y
Marker 2	🟪	M2	drdEdit	minSpanLengthSpacing	y
Marker 3	🟪	M2	drdEdit	minSpanLengthSpacing	y
Marker 4	🟪	M2	drdEdit	minSpanLengthSpacing	y
CDN20.M4.S.0 (SM) (1)(2)	☐				
M4 (2)	☐				
Marker 1	☐	M4	drdEdit	minSpacing	y
Marker 2	☐	M4	drdEdit	minSpacing	y
CDN20.M4.S.1 (DM) (1)(2)	☐				
M4 (2)	☐				
Marker 1	☐	M4	drdEdit	minSpacing	
Marker 2	☐	M4	drdEdit	minSpacing	
CDN20.M4.S.2.3 (SM) (1)(1)	🟪				
M4 (1)	🟪				
Marker 1	🟪	M4	drdEdit	minEndOfLineSpacing	y

4. Click an annotation in the Annotation Browser to highlight its marker in the workspace.

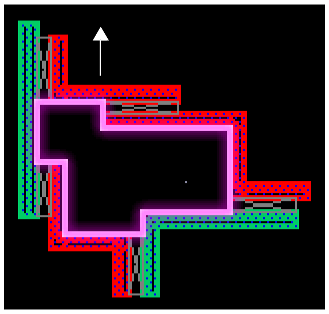


M4 (sameMask) minSpacing 0.07

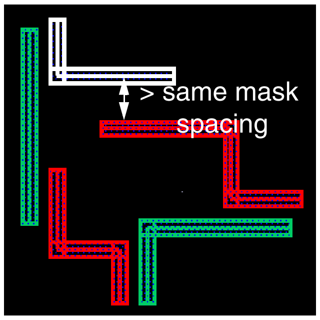
Fixing Multiple Patterning Violations

Move, stretch, and split functions use open space to move shapes away from each other to fix multiple patterning violations.

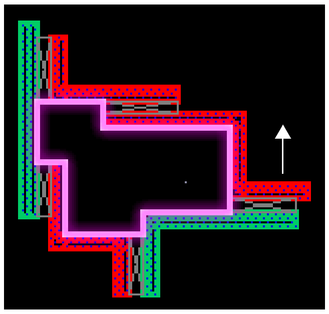
- **Move** shapes apart using *Edit – Move*.



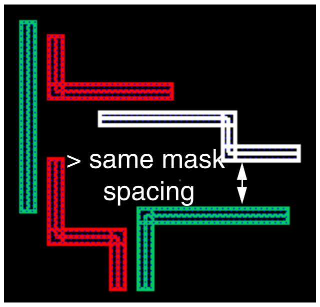
The top horizontal shape is moved north, leaving a gap to the next same color shape that is greater than same-mask spacing



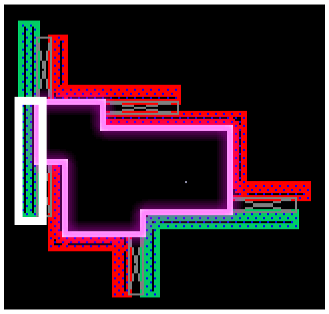
- **Stretch** shapes using *Edit – Stretch*.



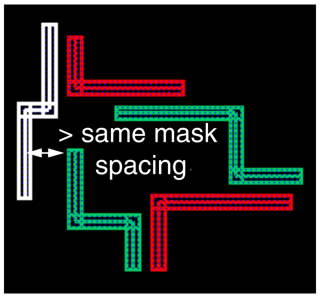
The bottom segment of the rightmost Z-shape is stretched north, leaving a gap to the next same color shape that is greater than same-mask spacing



- **Split** shapes using *Edit – Advanced – Split.*



The left-most shape is split with a shift of the lower section west, leaving a gap to the next same color shape that is greater than same-mask spacing



When shapes cannot be moved due to crowding, use **stitch** to replace one shape with two overlapping shapes, as described in Using Stitch for Multiple Patterning Violations.

Using Stitch for Multiple Patterning Violations

Stitch is used to replace one shape with two overlapping or *stitched* shapes on different masks of the same layer. This is useful to resolve multiple patterning violations when there is limited or no open space in the area of the violation.

The following topics are described in this section:

- Setting Stitch Constraints
- Creating Stitches
- Removing Stitches
- Stitch Restrictions

Setting Stitch Constraints

The `minStitchOverlap` constraint defines the minimum required overlap between two stitched shapes for a given layer. If this constraint is not specified, the minimum required overlap is equal to the `minWidth` constraint value.

The `minStitchOverlap` constraint can be set in the ASCII technology file. The following example sets the minimum required overlap to 0.06 user units for the `M1` layer in a constraint group named `minStitchOverlapCG`.

```
;( group                           )
;( ------------------- ------------ )
( "minStitchOverlapCG" nil nil 'or

  spacings(
   ( minStitchOverlap  "M1" 0.06 )
  ) ;spacings
) ;minStitchOverlapCG
```

The `minStitchOverlapCG` constraint group can be included as a member group of the `foundry` constraint group. This ensures that the constraints in `minStitchOverlapCG` are included in the constraint lookup hierarchy.

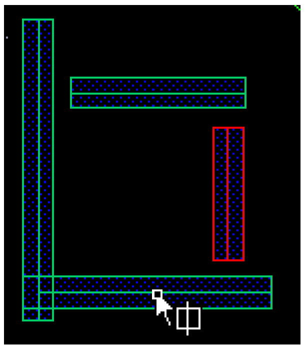
```
;( group                [override] )
;( --------   --------- )
( "foundry"        nil
              memberConstraintGroups(
              ; listed in order of precedence
              "minStitchOverlapCG"
              ) ;memberConstraintGroups
)
```

Creating Stitches

To create two stitched shapes from one shape,

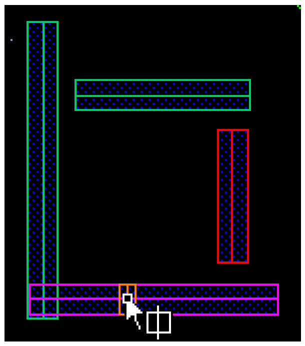
1. Enable stitching, as described in Using Stitch and UnStitch.
2. In the Multiple Patterning toolbar, click the *Stitch* icon.

   The command mode becomes `stitch` and the Stitch icon appears next to the pointer in the workspace.



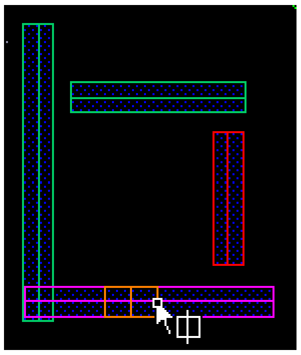
3. Click the shape to be stitched.

   The shape is highlighted in gray and the stitch overlap area is shown.



4. Move the cursor across the shape to the location for the stitch.

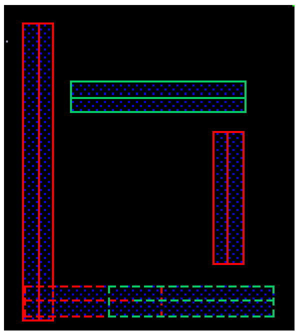
5. (Optional) Enlarge or shrink the stitch overlap area by pressing the `Ctrl` key while moving the pointer across the shape.



In this case, the `Ctrl` key was pressed while moving the pointer right to enlarge the stitch overlap area.

6. Click to make the stitch.

   The shape is broken into two shapes of opposite colors, forming a *different color group*, with the desired overlap.



Removing Stitches

To remove a stitch,

1. Enable unstitching, as described in Using Stitch and UnStitch.
2. In the Multiple Patterning toolbar, click the *UnStitch* icon.

   The command mode becomes `unstitch`, and the Stitch icon appears next to the mouse pointer.
3. Click a shape to unstitch it from its partner.

   The stitch is removed between the two shapes, forming one contiguous shape.

Stitch Restrictions

- Stitch will not operate on "gray" shapes with no color assignment.
- Stitched shapes cannot be stretched.
- Unstitch cannot be performed if a stitched shape has been deleted or moved so that it no longer overlaps the original stitch region with its partner.

Verifying the Consistency of Color Assignments

To ensure that constraints have been followed in the design, you must verify the consistency of color assignments between a schematic and layout. The two methods to verify the consistency of color assignments are described below:

- Virtuoso Coloring Check
- LVS Coloring Check

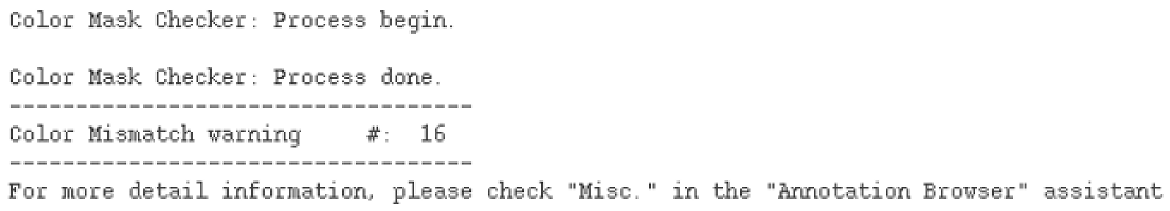
Virtuoso Coloring Check

To verify net mask color assignment using Virtuoso, use the *Net Color Constraint Check* and *CDF Color Check* options available in the *Verify* menu of VLS XL.

To verify net mask color assignment:

1. Choose *Connectivity – Check – Against Source*.
2. Type the *Library*, *Cell*, and *View* names of the schematic in the *Use schematic view* and *Use configuration view* sections in the Update Connectivity Reference form.
3. Click *OK*.
4. Choose *Verify – Net Color Constraint Check*.

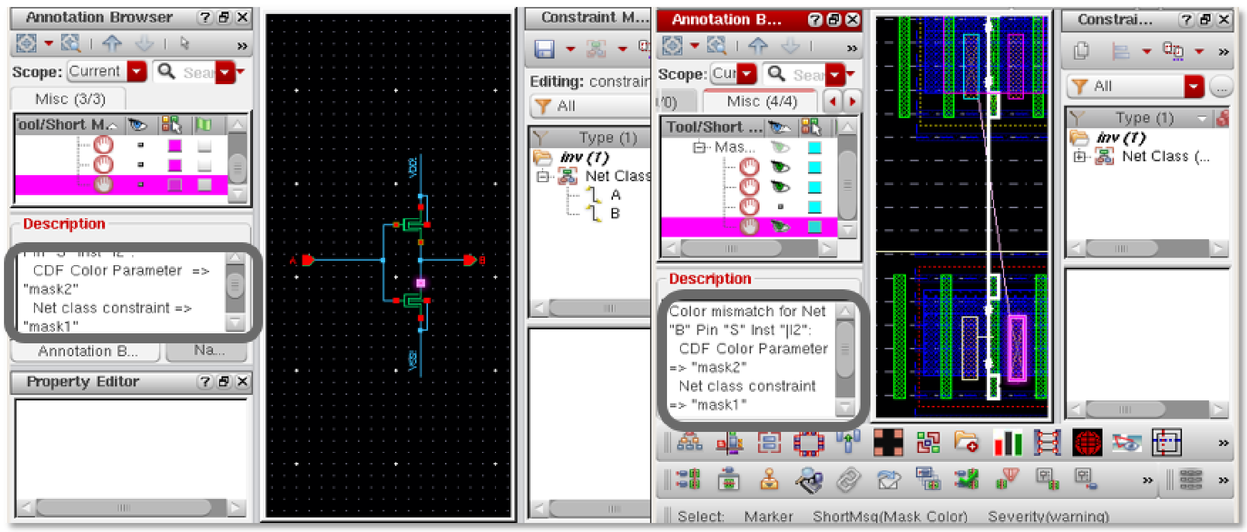
The details of the color mismatch and related warning are displayed in the CIW.



```
Color Mask Checker: Process begin.

Color Mask Checker: Process done.
-----------------------------------
Color Mismatch warning     #:  16
-----------------------------------
For more detail information, please check "Misc." in the "Annotation Browser" assistant
```

The mismatches are displayed in the Annotation Browser.



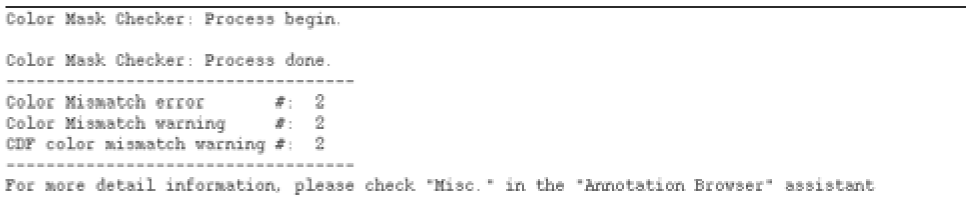
```
Pin "S" Inst "I2":
  CDF Color Parameter  =>
"mask2"
  Net class constraint =>
"mask1"
```

```
Color mismatch for Net
"B" Pin "S" Inst "|I2":
  CDF Color Parameter
=> "mask2"
  Net class constraint
=> "mask1"
```

To verify CDF color assignment:

1. Choose *Connectivity – Check – Against Source*.
2. Type the *Library*, *Cell*, and *View* names of the schematic in the *Use schematic view* and *Use configuration view* sections in the Update Connectivity Reference form.
3. Click *OK*.
4. Choose *Verify – CDF Color Check*.

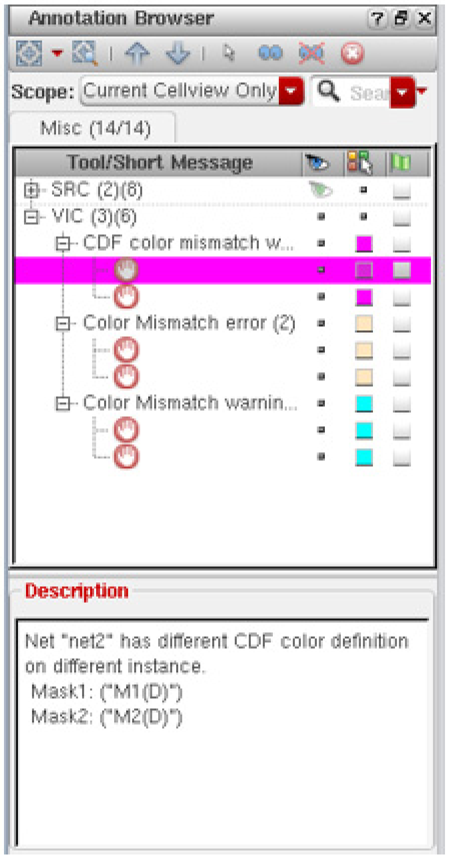
The details of the color mismatch and related warning are displayed in the CIW.



```
Color Mask Checker: Process begin.

Color Mask Checker: Process done.
------------------------------------
Color Mismatch error          #:  2
Color Mismatch warning        #:  2
CDF color mismatch warning #:  2
------------------------------------
For more detail information, please check "Misc." in the "Annotation Browser" assistant
```

The violation markers are displayed in the *Misc* tab of the Annotation Browser.



Net "net2" has different CDF color definition
on different instance.
Mask1: ("M1(D)")
Mask2: ("M2(D)")

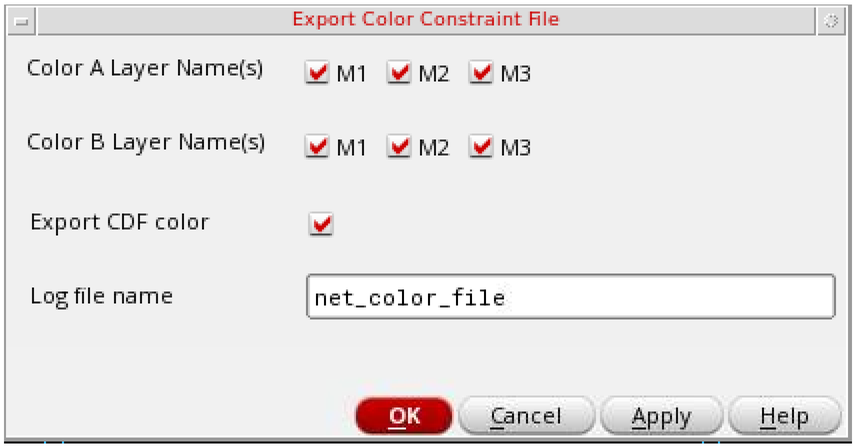
LVS Coloring Check

You can use Layout Versus Schematic (LVS) color checking to verify color assignments. You must generate the color constraint file to perform LVS coloring check. This file contains color information and is specified as an additional input to the LVS application.

To verify color assignment using LVS:

1. Choose *File – Export Color Constraint File* in the schematic view.

The Export Color Constraint File form appears.



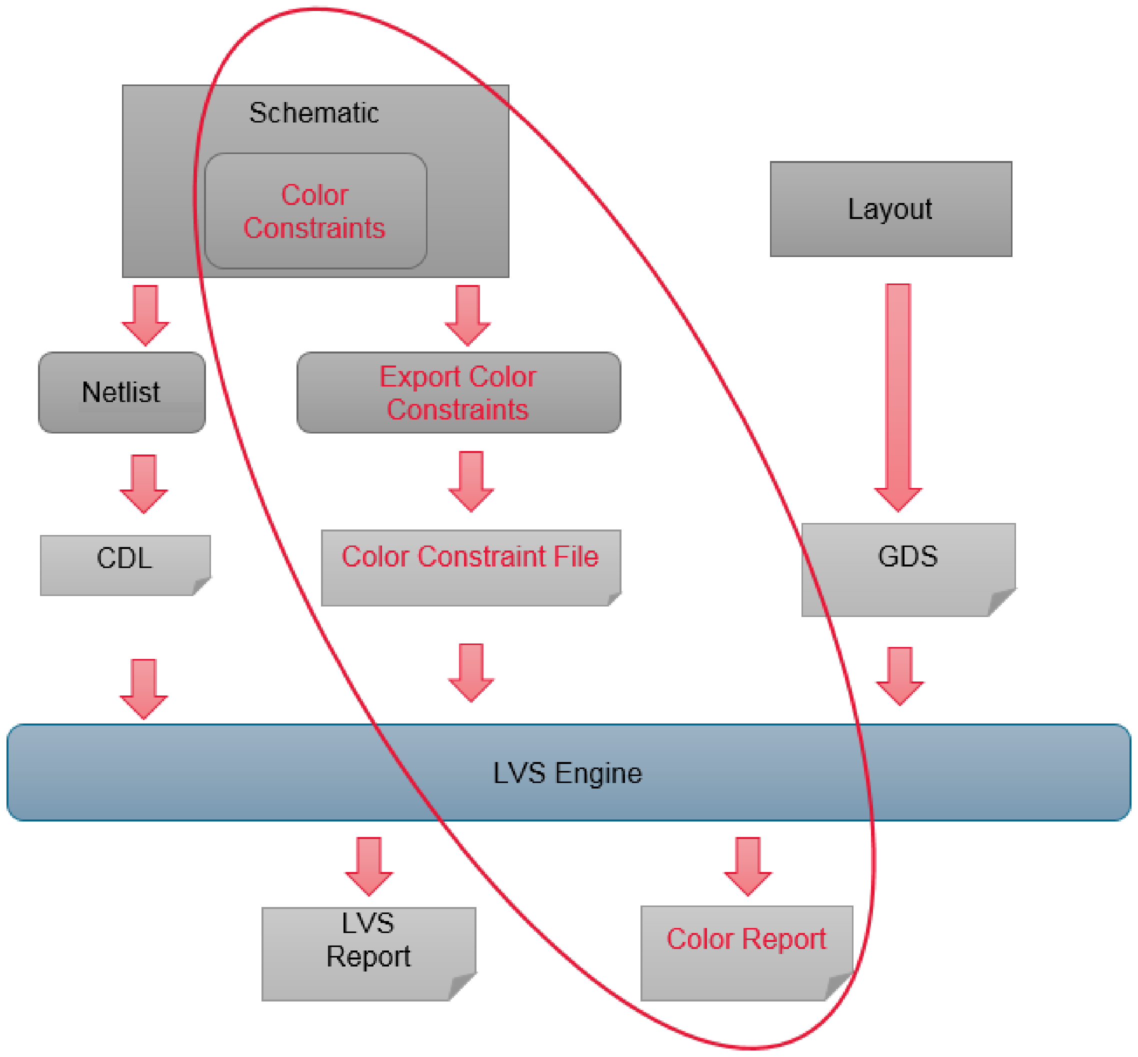
Export Color Constraint File

Color A Layer Name(s)	✓ M1 ✓ M2 ✓ M3
Color B Layer Name(s)	✓ M1 ✓ M2 ✓ M3
Export CDF color	✓
Log file name	net_color_file

2. Select the *Color A Layer Name(s)*, *Color B Layer Name(s)*, and *Export CDF color options* on the Export Color Constraint File form. Also, type the *Log file name*.

   Note: You can specify the values in the *Color A Layer Name(s)*, *Color B Layer Name(s)*, and *Log file name* fields using the environment variables, colorConstFileColorAName, colorConstFileColorBName, colorCDFCheck, and colorConstFileName.

The figure below displays the LVS-based coloring check flow.



6 Importing and Exporting Data

The following methods for importing and exporting data are discussed in this chapter:

- Using XStream
- Layer-Purpose Pair to Color Data
- Color Data Migration

Using XStream

The XStream translator is used to translate designs in Stream format to the OpenAccess database and conversely.

- XStream In translates designs in Stream format to the OpenAccess database.
- XStream Out translates designs from the OpenAccess database to Stream format.

This section describes the following:

- Exporting Stream Files with Coloring
- Importing Stream Files with Coloring
- Using Enhanced Layer Map Files for Coloring
- Types of Color Representations
- Handling Stitches
- Warning Messages from XStream

For more information on using XStream and the XStream GUI, refer to Just *Design Data Translator's Reference*.

Exporting Stream Files with Coloring

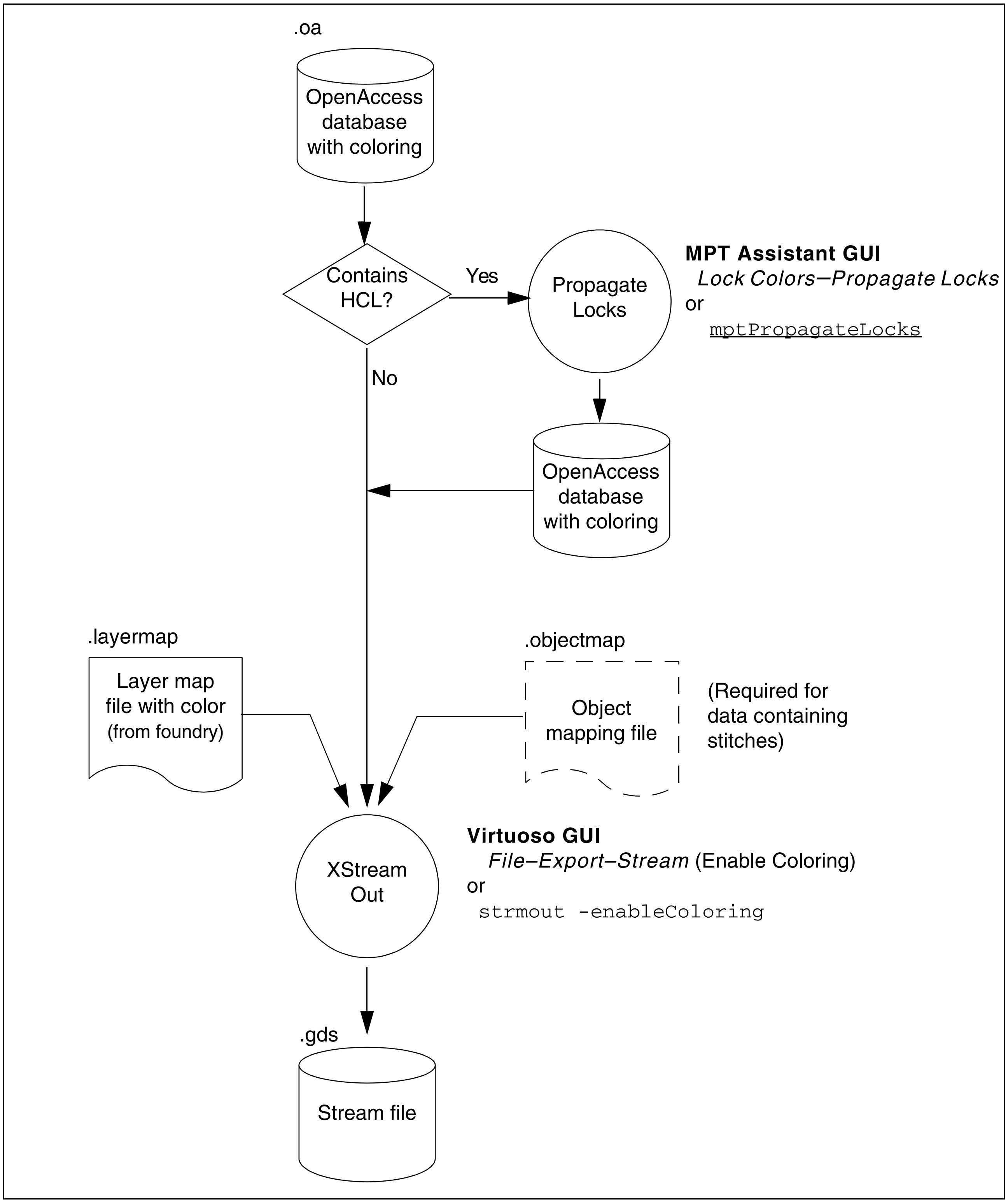
Figure 6-1 on page 140 shows the flow for translating an OpenAccess database with coloring to Stream (GDSII) format.

- A layer map is required that includes color mapping specified by the foundry. For more information, refer to Using Enhanced Layer Map Files for Coloring.
- If any shapes in your design are color locked using hierarchical color locking (HCL), follow the procedure in Propagating Locks to ensure that the HCL data is included during stream out. For more information on HCL, refer to Hierarchical Color Locking.
- If the OpenAccess database contains stitches, an object mapping file is also required, as described in Handling Stitches.
- The data is translated using the Virtuoso GUI (*File – Export – Stream* in the CIW), or by using `strmout` from the Textual User Interface.

Important

To stream out color, "Enable Coloring" must be enabled in the XStream Out GUI or "-enableColoring" must be included in the `strmout` command, before the layer map is loaded. Your layer map must include the photo mask color and color state columns.

Figure 6-1 Stream Out with Coloring Flow


.oa
OpenAccess database with coloring
Contains HCL?
Yes
No
Propagate Locks
MPT Assistant GUI
Lock Colors—Propagate Locks
or
mptPropagateLocks
OpenAccess database with coloring
.layermap
Layer map file with color (from foundry)
.objectmap
Object mapping file
(Required for data containing stitches)
XStream Out
Virtuoso GUI
File–Export–Stream (Enable Coloring)
or
strmout -enableColoring
.gds
Stream file

Importing Stream Files with Coloring

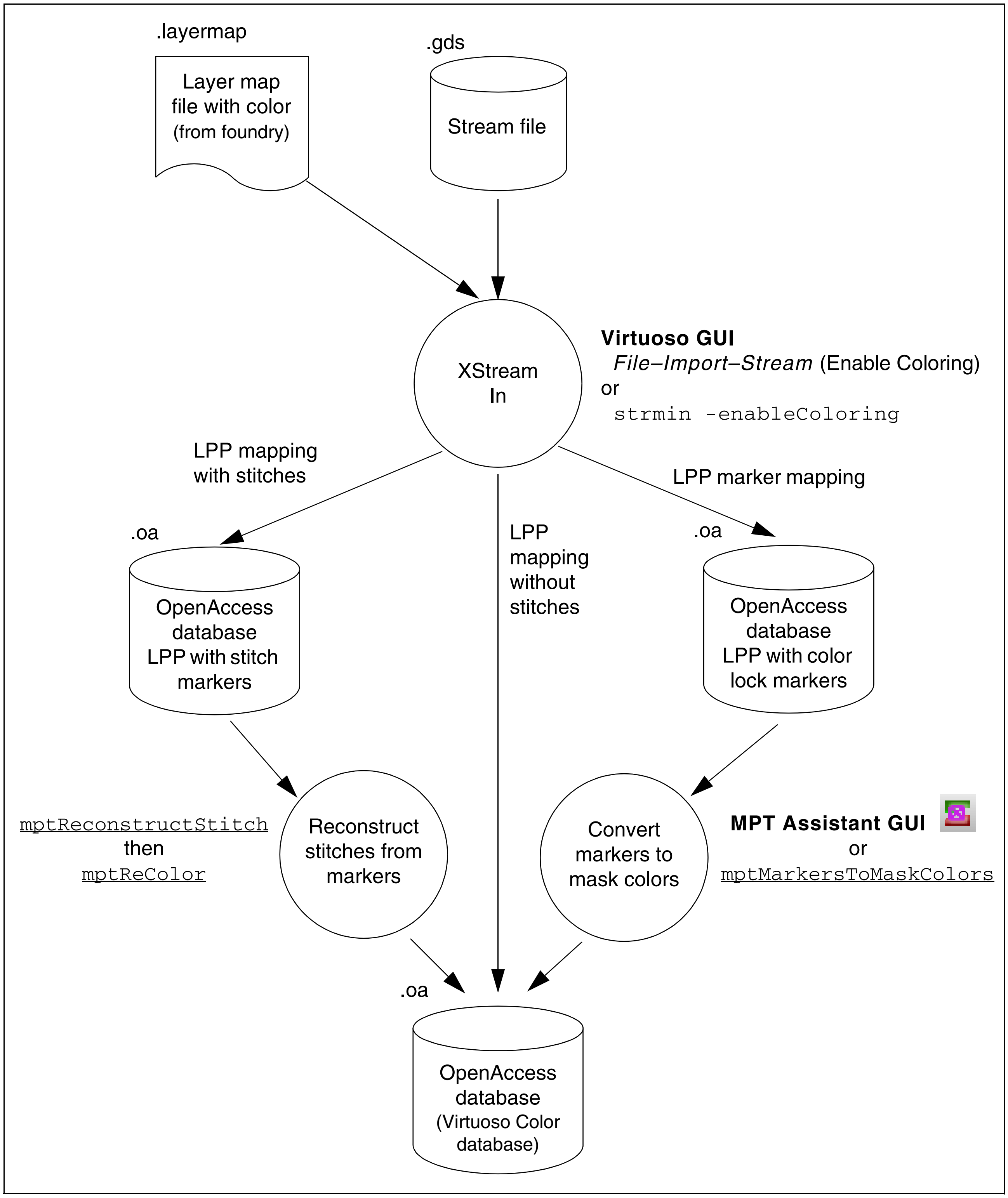
Figure 6-2 on page 142 shows the flow for translating a Stream (GDSII) file with coloring to an OpenAccess database.

- A layer map is required that includes the color mapping specified by the foundry. For more information, refer to Using Enhanced Layer Map Files for Coloring.
- The data is translated using the Virtuoso GUI (*File – Import – Stream*), or by using `strmin` from the Textual User Interface.
- If the resulting OpenAccess database includes additional markers to represent coloring, the markers must be converted to mask colors either by using the `mptMarkersToMaskColors` SKILL function or from the MPT Assistant.

Important

To stream in color, "Enable Coloring" must be enabled in the XStream In GUI or "-enableColoring" must be included in the `strmin` command, before the layer map is loaded. Your layer map must include the photo mask color and color state columns.

Figure 6-2 Stream In with Color Flow


.layermap
Layer map file with color (from foundry)
.gds
Stream file
XStream In
Virtuoso GUI
File–Import–Stream (Enable Coloring)
or
strmin -enableColoring
LPP mapping with stitches
LPP marker mapping
.oa
OpenAccess database LPP with stitch markers
LPP mapping without stitches
.oa
OpenAccess database LPP with color lock markers
mptReconstructStitch
then
mptReColor
Reconstruct stitches from markers
Convert markers to mask colors
MPT Assistant GUI
or
mptMarkersToMaskColors
.oa
OpenAccess database (Virtuoso Color database)

Using Enhanced Layer Map Files for Coloring

XStream can export and import color information to/from external tools that require layer-purpose pairs to represent mask colors. Two columns have been added to the layer map file for this purpose:

- **Photo mask color** with valid values `mask1Color` to `mask`*`X`*`Color` where *`X`* is the number of masks defined for the layer in the technology file.
- **Color State** with valid values `locked` and `unlocked`.

For example,

```
#layer purpose GDSlayer GDSdata material mask qual photomask  colorstate
Metal1 drawing    1        0
Metal1 drawing    1        1                       mask1Color unlocked
Metal1 drawing    1        2                       mask1Color locked
Metal1 drawing    1        3                       mask2Color unlocked
Metal1 drawing    1        4                       mask2Color locked
```

Both the locked and unlocked states of one color (for example, mask1Color) can be mapped to the same layer and data type in the stream (GDS) file.

Types of Color Representations

Stream file data is mapped to layer-purpose pairs in one of two ways, as determined by the foundry:

- Layer-Purpose Pair Mapping
- Layer-Purpose Pair Marker Mapping

Layer-Purpose Pair Mapping

This coloring maps GDS layer and data types to layer-purpose pairs. Each colored shape in GDS is represented by a shape in the OpenAccess database.

Layer-Purpose Pair Marker Mapping

This coloring maps GDS layer and data types to layer-purpose pairs. Each colored shape in GDS is represented in the OpenAccess database by two shapes, the original shape and a color marker. After this type of data is streamed in, the markers **must** be removed using the `mptMarkersToMaskColors` SKILL function or from the Multiple Patterning toolbar (described in Converting Markers to Mask Colors**) before** using the cluster-based color engine.

Handling Stitches

Stitches have special requirements when streaming out and streaming in.

Stream Out

When exporting an OpenAccess Database with stitches, you must provide an object mapping file for the stitches. This is a text file that maps the stitches by layer to the Stream layer and data type. For example,

```
#ObjectType ObjectSubType LayerName Stream# Datatype#
Stitch      None          Metal1    30      43
Stitch      None          Metal2    34      43
Stitch      None          Metal3    38      43
```

The object mapping file can be specified using the `-objectMap` command-line option to `strmout`, or from the StreamOut Options form's Object tab GUI.

Stream In

After importing a stream file that contains stitches, stitches will appear as stitch shape markers that identify the overlap area. You **must** reconstruct the stitches before using the cluster-based color engine. Use the `mptReconstructStitch` SKILL function for each layer with stitches, then `mptReColor`.

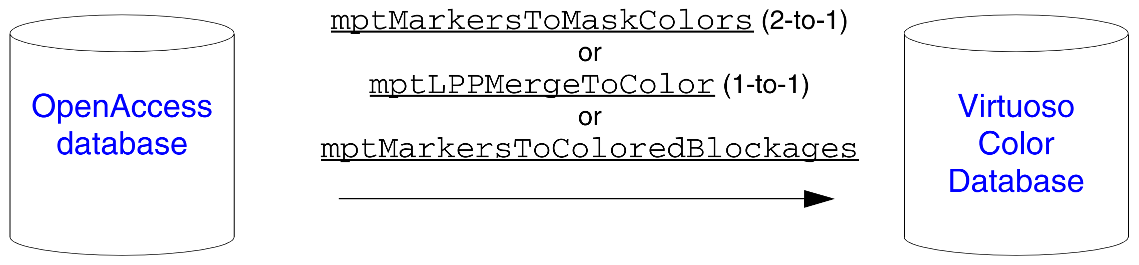
Warning Messages from XStream

XSTRM-325

If a colored shape is found without a color-aware mapping in the layer map file, a warning message indicates the layer-purpose pair for the shape with its coordinates, photomask color, and color state. This can be intentionally done to filter out shapes on specific layer-purpose pairs with a specific photomask color and/or color state. For example, if you could exclude a mapping for 'metal1:drawing' with photomask 'mask2Color' and color state 'unlocked', any shapes of the excluded mapping type would not be streamed out.

Layer-Purpose Pair to Color Data

Some designs use separate layer-purpose pairs to represent the multiple patterning mask information. SKILL functions are provided to convert that data to a database that can be used by the Virtuoso color engine.


OpenAccess database
mptMarkersToMaskColors (2-to-1)
or
mptLPPMergeToColor (1-to-1)
or
mptMarkersToColoredBlockages
Virtuoso Color Database

Merging Layer-Purpose Pairs on Two Layers (2-to-1)

For designs that represent colored data by overlapping the metal shapes on one layer-purpose pair with precoloring markers on another layer-purpose pair, use `mptMarkersToMaskColors`. From the layout view, the marker and drawing layer will be merged into one drawing layer through the hierarchy.

Note: If XStream is used to translate data of this type, the precoloring markers will be converted but the covered metal layer shapes will remain.

For details on using this function, refer to `mptMarkersToMaskColors` in *Virtuoso Layout Suite SKILL Reference*.

Merging Multiple Layer-Purposes on One Layer (1-to-1)

For designs that represent colored data using multiple purposes on one layer, use `mptLPPMergeToColor`. This function creates a new design hierarchy in which the shapes in the multiple layer-purpose pairs are merged to shapes on a single layer-purpose pair. The shapes in the merged representation will have colors corresponding to the layer-purpose pair in the original layer-purpose pair representation.

Output can be to the same cellview (overwrite), a different view in the same library, the same view in a different library, or a new library and view.

For details on using this function, refer to `mptLPPMergeToColor` in *Virtuoso Layout Suite SKILL Reference*.

Transforming Shapes on Specified Purposes to Colored Blockages

For designs that represent colored blockages using shapes on specific layer-purposes, use `mptMarkersToColoredBlockages`. From the layout view, the shapes on those purposes are transformed to colored blockages on the same layer through the hierarchy.

Color Data Migration

This section describes how data is handled when migrating from a previous release or another product version.

- From/To IC6.1.x
- To ICADV12.2 LA4 or above
- To ICADV12.2 Base Release or ICADV12.2 ISR1

The standalone utility, mptScan, can be used to detect and correct coloring issues, and remove coloring information from a design.

From/To IC6.1.x

IC6.1.x can open ICADV12.2 layouts without any coloring information in read or edit mode.

IC6.1.x can open ICADV12.2 layouts with coloring information in read-only mode, with the following exceptions: Virtuoso Space-based Router and Virtuoso Routing IDE.

To ICADV12.2 LA4 or above

Important

The colorSpec bits issue that is described in this section has been resolved in ICADV12.2 LA4 ISR2. If you are running Virtuoso ICADV12.2 LA4 ISR2, you do not need to run mptScan to check for or clear colorSpec bits.

Some objects that were colored in a previous version of ICADV12.2 appear gray or uncolored in ICADV12.2 LA4 and ICADV12.2 LA4 ISR1. This was due to data and/or libraries using outdated colorSpec bits that are not supported. ICADV12.2 LA4 was enhanced to support up to eight colors in the database, using some bits that had been used for colorSpec in early releases of ICADV12.1.

To ensure that the colorSpec bits are not set in a design,

1. Run mptScan at the command prompt to scan for colorSpec bits in the data.
   ```
   mptScan -lib libName -report colorSpec
   ```
2. If colorSpec bits are detected in step 1, run mptScan to clear the colorSpec bits.
   ```
   mptScan -lib libName -clear colorSpec
   ```
3. Open the design in Virtuoso to verify that the coloring is correct.

To ICADV12.2 Base Release or ICADV12.2 ISR1

Modifying designs in ICADV12.2 base release and ICADV12.2 ISR1 can result in the data being read-only in ICADV12.1 and pre-LA6 versions of ICADV12.2.

If you do not color or lock geometry, modifying a via with the Property Editor in ICADV12.2 base release or ICADV12.2 ISR1 will result in setting the coloring infrastructure to a version that can be edited only by ICADV12.2 LA6 and later releases. If you are coloring or locking geometry currently using ICADV12.2 LA4 and earlier versions, using ICADV12.2 base release to edit vias or perform coloring operations such as lock propagation will result in using the LA6 version of the coloring infrastructure. Any design with the ICADV12.2 LA6 version of the coloring infrastructure can be read into ICADV12.1 and ICADV12.2 LA4 and earlier releases, but cannot be edited.

Converting Colored Data to Be Editable in Earlier Releases

To determine if a design is using the ICADV12.2 LA6 version of the coloring infrastructure, run the mptScan utility with the `-checkColorVer` argument:

```
mptScan -lib <libName> -checkColorVer
```

It will report the MPToolkit coloring data version. For example,

```
INFO: Cellview "mylib/mycell/layout" contains MPToolkit coloring data version: 4,
CDS coloring data version:3
INFO: MPToolkit coloring data version can be reduced from 4 to 1.
```

In this case, no partially locked vias were found and fewer than four mask colors were used, so the MPToolkit coloring data version can be reduced from 4 to 1. If four mask colors were used with no partially locked vias, the MPToolkit coloring data version could be reduced from 4 to 3.

The MPToolkit coloring data versions are described in the following table:

MPToolkit coloring data version	Description	Product Version
`1`	Supports up to 3 colors	ICADV12.1 (supported 2 colors), ICADV12.2 before LA4
`2`	Supports 4+ colors	ICADV12.2 LA4, ICADV12.2 LA4 ISR1
`3`	Supports 4+ colors without colorSpec conflict	ICADV12.2 after LA4 ISR1

MPToolkit coloring data version	Description	Product Version
`4`	Supports 4+ colors and partial via locking	ICADV12.2 LA6
`5`	Supports 4+ colors, partial via locking, and fixed mask layer shifting on instances	ICADV12.2 ISR2

Note: Any MPToolkit coloring data version above 1 cannot be edited in ICADV12.1.

To revert a design without colored and locked vias to the pre-ICADV12.2 LA6 version of the coloring infrastructure, run the mptScan utility with the `-reduceColorVer` argument:

```
mptScan -lib <libName> -reduceColorVer
```

It will report the version reduction. For example,

```
INFO: MPToolkit coloring data version was reduced from 4 to 1.
```

The version reduction preserves the coloring information and enables the data to be edited in earlier product versions that support the new MPToolkit coloring data version.

Partially locking a via through the Property Editor or lock propagation will cause the data to be converted to the ICADV12.2 LA6 infrastructure. The converted data cannot be reverted to the previous coloring data version unless the designs are changed to fully lock or fully unlock all vias. Hence, it is recommended that if you use color and color locking, ensure that any design groups that move to ICADV12.2 base release or later do not share data with design groups that use limited access releases of ICADV12.2 prior to LA6. If this is not possible, contact Cadence Customer Support for further assistance.

mptScan

`mptScan` is a standalone utility for detecting and correcting coloring issues, and for clearing the coloring information in a design. It can be accessed from `*installDirPath*/tools/bin`.

Important

Cadence strongly recommends that the latest available `mptScan` be run on an entire library so that the data is consistent for all the cells in the library.

Syntax

```
mptScan -lib libName [options]
```

Required Arguments

`-lib` *`libName`*	Processes cellviews in this library.

Optional Arguments

`-libPath` *`path`*	Specifies the full path to the library.
`-cell` *`cellName`*	Processes only cellviews in the library with this cell name.
`-view` *`viewName`*	Processes only cellviews in the library with this view name. The cell (`-cell`) must also be specified.
`-copyTo` *`path`*	Copies the library to the specified path before processing.
`-clear colorSpec`	Clears the colorSpec bits in the design. Cannot be used with `-removeData`, `-repair`, or `-report` arguments.
`-report colorSpec`	Scans the design for color data with set colorSpec bits. Cannot be used with `-clear`, `-removeData`, or `-repair` arguments.
`-removeData`	Removes the advanced node coloring information from the design. Cannot be used with `-clear`, `-repair`, or `-report` arguments.

`-repair`	Repairs data inconsistency issues found in the advanced node coloring data. Cannot be used with `-clear`, `-removeData`, or `-report` arguments.
`-checkColorVer`	Reports the current coloring data version for each cellview and whether it can be reduced. Cannot be used with `-clear`, `-reduceColorVer`, `-removeData`, `-repair`, or `-report` arguments.
`-reduceColorVer`	Reduces the current coloring data version to the lowest possible value that supports the coloring (MPT) features in the cellview. The version reduction preserves coloring information and enables the data to be edited in earlier product versions that support the new coloring data version. Cannot be used with `-clear`, `-checkColorVer`, `-removeData`, `-repair`, or `-report` arguments.
`-h, -help`	Prints the usage message.
`-logFile` *`logfilename`*	Specifies the output log file. Default: `mptScan.log`
`-noInfo` *`msgIds`*	Prevents the specified INFO message IDs from being output.
`-noWarning` *`msgIds`*	Prevents the specified WARNING message IDs from being output.
`-templateFile` *`file`*	Uses the specified file containing the arguments to be used when running this command.
`-v`	Prints the tool, format, and library information.
`-verbose`	Outputs additional information during processing.
`-version`	Prints the tool and format version information.

Examples

```
mptScan -lib myLib
```

```
WARNING: (mptScan-2): Repair needed for shape cluster data that references a
cluster on a different layer.
WARNING: (mptScan-2): Repair needed for via cluster data that references a
cluster on a different layer.
INFO: Found advanced node coloring and 2 data issues in 'myLib/top/layout_bk'.
INFO: Found advanced node coloring in 1 (out of 1) layouts.
Finished: mptScan
```

Scans the cellviews in the `myLib` library for advanced node coloring issues and outputs warnings for the issues that are found, along with a summary for each scanned cellview.

```
mptScan -lib myLib -repair
```

```
INFO: (mptScan-2): Repaired shape cluster data that referenced a cluster on a
different layer. Verify that coloring in this design is correct. If not correct,
recolor the design.
INFO: (mptScan-2): Repaired via cluster data that referenced a cluster on a
different layer. Verify that coloring in this design is correct. If not correct,
recolor the design.
INFO: Repaired 2 found issues in 'myLib/top/layout_bk'.
INFO: Repaired advanced node coloring data issues in 1 (out of 1) layouts.
Finished: mptScan
```

Repairs advanced node coloring issues in the `myLib` library.

```
mptScan -lib HCL -removeData
```

```
INFO: Removed advanced node coloring from 'HCL/test/layout'.
INFO: Removed advanced node coloring from 1 (out of 1) layouts.
Finished: mptScan
```

Removes the coloring information from the cellviews in the `HCL` library.

```
mptScan -lib myLib -report colorSpec
```

```
Running: mptScan -logFile mptScan.log -lib myLib -report colorSpec
INFO: Found colorSpec data in 'myLib/top/layout_bk'.
INFO: Found colorSpec data in 1 (out of 12) layouts.
Finished: mptScan
```

Scans the cellviews in the `myLib` library for the set colorSpec bits.

```
mptScan -lib myLib -clear colorSpec
```

```
Running: mptScan -logFile mptScan.log -lib myLib -report colorSpec
INFO: Cleared colorSpec data in 'myLib/top/layout_bk'.
INFO: Cleared colorSpec data in 1 (out of 12) layouts.
Finished: mptScan
```

Clears the colorSpec bits in the `mylib` library.

```
mptScan -lib myLib -checkColorVer
Running: mptScan -logFile mptScan.log -lib myLib -checkColorVer
INFO: Cellview "myLib/mycell/layout" contains MPToolkit coloring data version:
4, CDS coloring data version:3
INFO: MPToolkit coloring data version can be reduced from 4 to 1.
```

Reports the coloring data versions for the cellviews in the `myLib` library and whether the MPToolkit coloring data version can be reduced.

```
mptScan -lib myLib -reduceColorVer
Running: mptScan -logFile mptScan.log -lib myLib -reduceColorVer
INFO: Cellview "myLib/mycell/layout" contains MPToolkit coloring data version:
4, CDS coloring data version:3
INFO: MPToolkit coloring data version was reduced from 4 to 1.
```

Reduces the coloring data versions for the cellviews in the `myLib` library, if possible. The version reduction preserves the coloring information so that it can be edited using an earlier product version that supports the new coloring data version.

7

Fully Colored Backannotation Flow

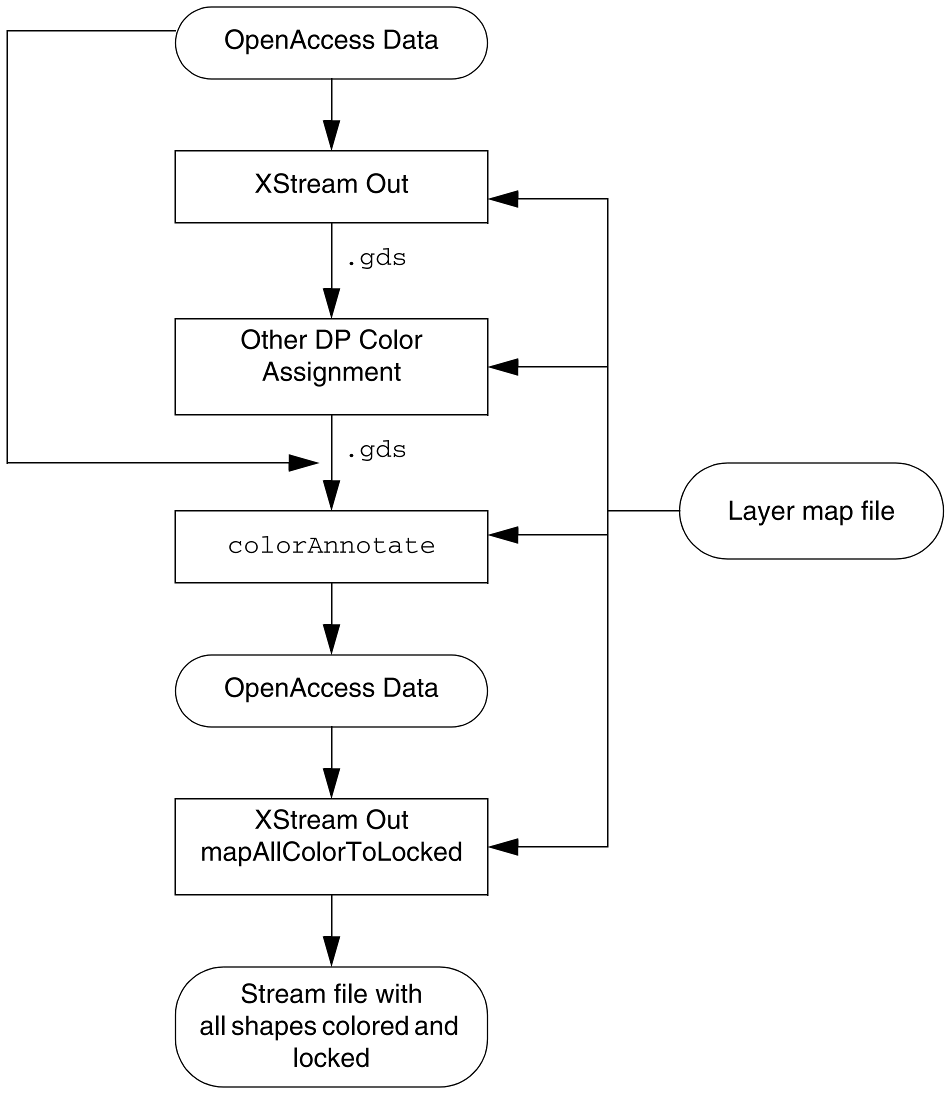
In the Fully Colored Backannotation flow, the shapes and nets in a design that were manually color locked will retain their color and locked state. Shapes that are not color locked or are gray will be colored using a third-party engine for decomposition and a `colorAnnotate` utility to backannotate color.

This chapter covers the following:

- Overview
- Enhancements in the Layer Map File Format
- Using colorAnnotate

Overview

The Fully Colored Backannotation flow is shown in the following figure.


OpenAccess Data
XStream Out
.gds
Other DP Color Assignment
.gds
Layer map file
colorAnnotate
OpenAccess Data
XStream Out mapAllColorToLocked
Stream file with all shapes colored and locked

The steps in this flow are described in the following sections:

- Creating the Layer Map File
- Translating Data to a Stream File
- Assigning Colors to Unlocked and Uncolored Shapes Using a Third-Party Color Engine
- Merging Locked Shapes with Shapes Colored by the Third-Party Color Engine

- Creating a Stream File with All Colors Locked

Creating the Layer Map File

To use the Fully Colored Backannotation flow, you must have a layer map file that maps the data to color annotations using the `Color Annotate` field. The same layer map file can be used for all the steps in the flow. For more information, see Enhancements in the Layer Map File Format.

Translating Data to a Stream File

Use XStream Out to translate the data to a Stream file (`.gds`) for input to the third-party color engine. For more information, refer to *Design Data Translator's Reference*

Assigning Colors to Unlocked and Uncolored Shapes Using a Third-Party Color Engine

Use the third-party tool to assign colors to all the shapes that are not color locked in the database, including gray (uncolored) shapes. During this process, previously colored shapes can change colors. The third-party tool outputs a Stream file that consists of color annotations only for the shapes that were colored by the third-party tool.

Merging Locked Shapes with Shapes Colored by the Third-Party Color Engine

Use the new `colorAnnotate` utility to merge the original data with the color annotations in the third-party tool Stream file output. For more information, see Using colorAnnotate. The color-locked shapes in the original data are not changed. All other shapes are colored according to the color assignments from the third-party tool. At the end of this step, all shapes will be colored.

Creating a Stream File with All Colors Locked

Use XStream Out with the new *Map All Colors to Locked* option to create a Stream file in which all shapes are colored and locked. For information about using the XStream Out GUI, see New Map All Colors to Locked Option in XStream Out.

Enhancements in the Layer Map File Format

A Color Annotate field has been added to the layer map file syntax and is mutually exclusive with the Color State field. It is used by the third-party color engine to output annotations to a Stream file, and by the `colorAnnotate` utility to assign colors to shapes in the OpenAccess database using the annotations in the Stream file.

```
<Layer Name>  <Purpose Name>  <Stream Layer Number>  <Stream
DataType>   [Material Type]  [Mask Number]  [Qualifier]
[Photomask Color]  [Color State] | [Color Annotate]
```

where:

- Columns 1 to 4 are mandatory.
- Columns 5 to 9 are optional.
- Allowed combinations from columns 5 to 9 are:
    - * `<Material Type>`
    - * `<Material Type>` `<MaskNumber>`
    - * `<Material Type>` `<MaskNumber>` `<Qualifier>`
    - * `<Material Type>` `<Qualifier>`
    - * `<Qualifier>`
    - * `<Material Type>` `<PhotoMaskColor>` `<ColorState>`
    - * `<Material Type>` `<MaskNumber>` `<PhotoMaskColor>` `<ColorState>`
    - * `<Material Type>` `<MaskNumber>` `<Qualifier>` `<PhotoMaskColor>` `<ColorState>`
    - * `<Material Type>` `<Qualifier>` `<PhotoMaskColor>` `<ColorState>`
    - * `<Qualifier>` `<PhotoMaskColor>` `<ColorState>`
    - `<PhotoMaskColor>` `<ColorAnnotate>`
- Valid values for the `Qualifier` column are `Pin`, `floating`, and `cutSize`:<*`float`*>:<*`float`*>.
- Valid values for the `Photomask Color` column are `mask1Color` to `mask`*`X`*`Color` where *`X`* is the number of masks defined for the layer in the technology file.
- Valid values for the `Color State` column are `locked` and `unlocked`.

- Valid value for the `Color Annotate` column is `colorAnnotate`. Lines with this value will be ignored by XStream In and XStream Out.
- `Color State` and `Color Annotate` cannot be used in the same line.

Note: Material type and Mask number will be considered during XStream In and will be ignored during XStream Out.

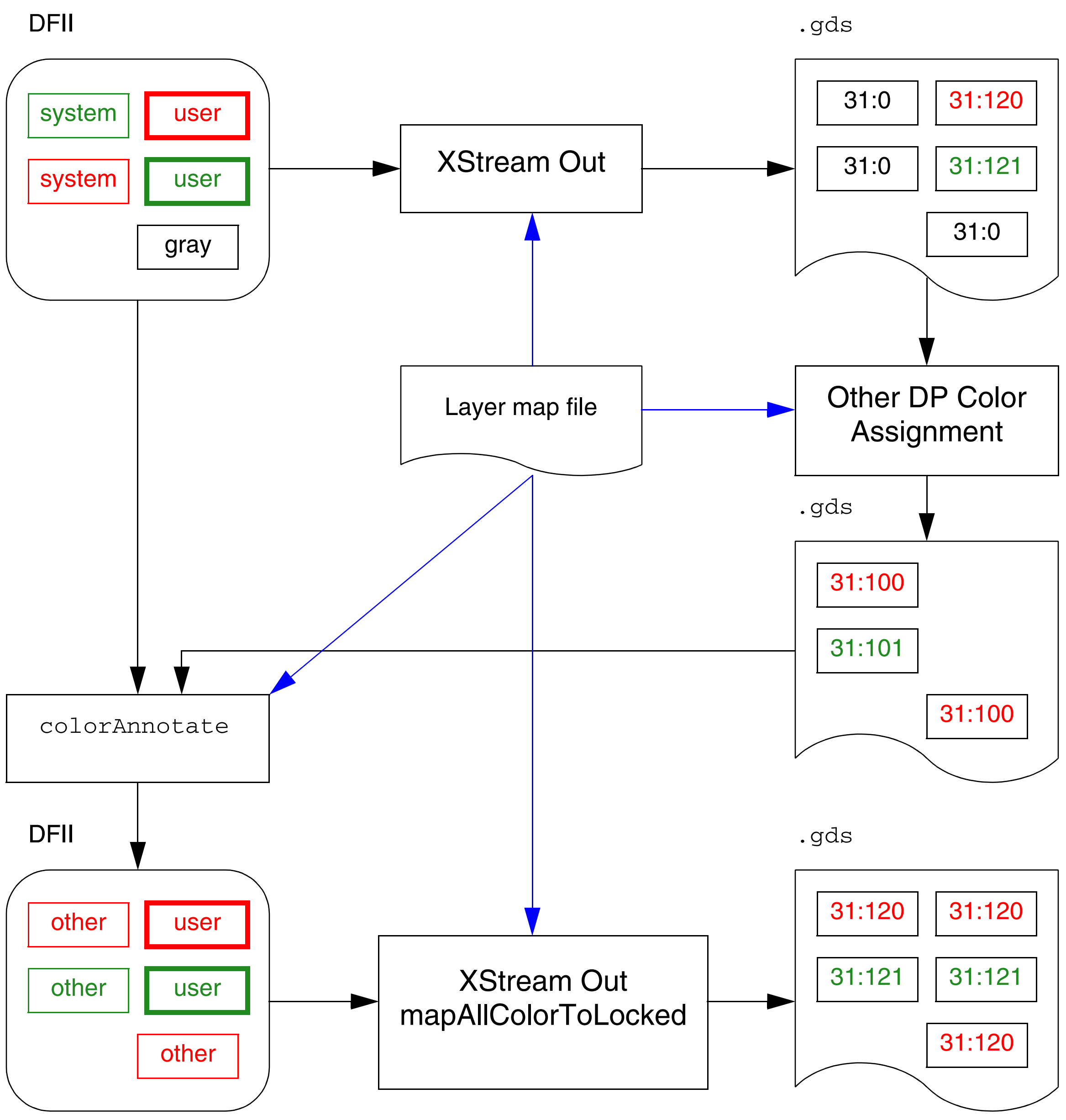
* `Qualifier`, `floating` is only valid for purpose name, `fill` and `fillOPC`.

Sample Layer Map File for Fully Colored Backannotation Flow

The following is an example of a layer map file for the Fully Colored Backannotation flow:

```
# DFII   DFII           Stream    Stream
# Layer  Layer-Purpose  Layer #   Data Type  MaskColor  ColorState/ColorAnnotate
Metal1   drawing        31        0
Metal1   drawing        31        200        mask1Color locked
Metal1   drawing        31        201        mask2Color locked
Metal1   drawing        31        100        mask1Color colorAnnotate
Metal1   drawing        31        101        mask2Color colorAnnotate
Metal2   drawing        32        0
Metal2   drawing        32        200        mask1Color locked
Metal2   drawing        32        201        mask2Color locked
Metal2   drawing        32        100        mask1Color colorAnnotate
Metal2   drawing        32        101        mask2Color colorAnnotate
```

The following figure shows the progression of sample data using this layer map in the Fully Colored Backannotation flow.



Using colorAnnotate

The `colorAnnotate` utility merges color annotations in a Stream file with the data in an OpenAccess database, including the data at the specified top-level cell and the hierarchy below it.

Important

The OpenAccess library and the Stream file (`.gds`) should have the same hierarchical structure. The only difference can be the flattening of instances in the Stream file. Other hierarchy modifications (for example, if the mosaic in the OpenAccess database is represented as a set of different instances in the Stream file), can cause `colorAnnotate` to work incorrectly.

`colorAnnotate` is included in the installation and can be called from:

```
<install>/bin/colorAnnotate
```

You can run `colorAnnotate` from the command line.

```
colorAnnotate -library libName -strmFile fileList -topCell cellName [options]
```

Required Arguments

`-library` *`libName`*	Name of the library on which shape decomposition and color assignment was performed.
`-strmFile` *`fileList`*	Comma-separated list of input color Stream files.
`-topCell` *`cellName`*	Name of the top-level cell

Optional Arguments

`-cellMap` *`cmfileName`*	Name of the cell map file. This is an ASCII file that stores the mapping information between the OpenAccess cellview names and the Stream cell names.
`-command` *`shellCommand`*	Runs *`shellCommand`* before importing and merging the color annotations in the Stream file.
`-createColorsOnTopLevel`	Creates all hierarchical color annotations in the top cell specified by the `-topCell` argument.
`-layerMap` *`lmfileName`*	Name of the layer map file. This an ASCII file that maps the layer numbers and data types used in the Stream file to layer-purpose pairs in OpenAccess, and vice-versa. The default is `layers.map`.
`-logFile` *`lfileName`*	Name of the log file to record the translation process steps and the messages generated by `colorAnnotate`.
`-noInfo` *`msgIDs`*	Suppresses the specified INFO messages. *`msgIds`* is a space-separated list of numbers. Each number in the list represents the numerical portion of the ID for the message to be suppressed. 0 is not a valid message number. Suppressed messages do not appear on the terminal or in the log file.

`-noWarning` *`msgIDs`*	Suppresses the specified WARNING messages. *`msgIds`* is a space-separated list of numbers. Each number in the list represents the numerical portion of the ID for the message to be suppressed. 0 is not a valid message number. Suppressed messages do not appear on the terminal or in the log file and are not included in the total of WARNING messages displayed in the summary.
`-numThreads` *`count`*	Number of threads to use for translating.
`-templateFile` *`file`*	Name of the template file. This ASCII file specifies the arguments for the command.
`-view` *`viewName`*	Name of the view to translate. The default is `layout`.
`-h | -help`	Prints the help message.
`-v`	Prints the tool, format, and library version information.
`-version`	Prints the tool and format version information.

Example

```
colorAnnotate -library Test -topCell Test1 -strmFile Test1_DP.gds
-layerMap showbug.map -logFile showbug.log
```

Merges data in `Test/Test1/layout` with color annotations in `Test1_DP.gds`, using `colorAnnotate` mappings in the `showbug.map` layer map file. Messages are output to `showbug.log`.

A

MPT Environment Variables

This chapter describes environment variables used with Virtuoso® Multi-Patterning Technology (MPT) and the SKILL functions that are used to set and check environment variable values.

Note: Only the environment variables documented in this chapter are supported for public use. All other MPT-related environment variables, regardless of their name or prefix, and undocumented aspects of the environment variables described below, are private and are subject to change at any time.

Related Topics

- Environment Variable Functions
- List of Virtuoso MPT Environment Variables

Environment Variable Functions

The following environment variable functions are described in this section:

- Checking the Value of an Environment Variable
- Setting the Value of an Environment Variable

Checking the Value of an Environment Variable

To determine the current value of an environment variable,

➤ Type in the CIW,

```
envGetVal( t_tool t_varName )
```

where

t_tool	Is the name of the tool.
t_varName	Is the name of the environment variable.

For example,

```
envGetVal( "mpt" "coloringEngineEnabled" )
```

Setting the Value of an Environment Variable

To set an environment variable,

➤ Type in the CIW,

```
envSetVal( t_tool t_varName s_varType g_newVal )
```

where

t_tool	Is the name of the tool.
t_varName	Is the name of the environment variable.
s_varType	Is the value type for the environment variable.
g_newVal	Is the value to be assigned to the environment variable.

For example,

```
    envSetVal( "mpt" "coloringEngineEnabled" 'boolean t )
```

The environment variables can also be set in your `.cdsinit` or `.cdsenv` file and are described in the following section, with examples.

List of Virtuoso MPT Environment Variables

These `mpt` environment variables are used by Virtuoso MPT.

- allowLockShiftOverride
- autoPropagateLock
- checkHCLOverUnlock
- colorCDFCheck
- colorCDFParamPrefix
- colorConstFileColorAName
- colorConstFileColorBName
- colorConstFileName
- coloredPurposeTypes
- coloringEngineEnabled
- coloringFilterSize
- colorShiftingLayers
- complianceCheckerLimit
- complianceCheckerReport
- defaultColoringMethod
- deleteConnectedShapes
- displayMaskColor
- displayMaskColorExcludeViaLayers
- displayMaskColorMode
- displaySystemColor
- dontColorPCells
- drawSurroundingOn
- enableHCLCreation
- enableHCLCreationOnPcells

- enableMarkersToMaskColors
- explicitColoredPurposes
- extractorStopLevel
- forcePcellRecolorOnEval
- forceRandomOnLayerType
- globalColorShiftingPolicy
- globalColorShiftingPolicyForPcells
- hideStitchingTools
- lockAllHCLPolicy
- mergeColoredPacket
- mergeColoredPacket
- mptConstraintGroup
- onlyCheckActiveWSP
- overrideLockOnConnectedShapes
- propagateLocksToConnectedShapes
- propagateAnySameMaskState
- pvsDeckFile
- pvsLayerMapFile
- reColorGUIScope
- reColorReadOnlyCellView
- shiftCutColorFromToolbarButton
- showCDFchecks
- trackColoringOnlySnappedShapes
- unclusteredShapeColor
- updateColorOnActivate

This `cdba` environment variable is used by Virtuoso MPT:

- copyMPAttributes

allowLockShiftOverride

```
mpt allowLockShiftOverride boolean { t | nil }
```

Description

Specifies whether color-locked shapes can be color-shifted using the Multiple Patterning toolbar. The default is `nil`.

GUI Equivalent

Command	*Options – Multiple Patterning*
Field	*Allow shifting of locked shapes* (Multiple Patterning Options)

Examples

```
envGetVal("mpt" "allowLockShiftOverride")
envSetVal("mpt" "allowLockShiftOverride" 'boolean t)
envSetVal("mpt" "allowLockShiftOverride" 'boolean nil)
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

autoPropagateLock

```
mpt autoPropagateLock boolean { t | nil }
```

Description

Specifies whether locks will be automatically propagated to connected shapes on the same layer when the lock is initiated by editing (for example, by adding a shape, moving, stretching). The default is `nil`, color locking operates only on the selected shape.

GUI Equivalent

Command	*Options – Multiple Patterning*
Field	*Propagate locks while editing* (Multiple Patterning Options)

Examples

```
envGetVal("mpt" "autoPropagateLock")
envSetVal("mpt" "autoPropagateLock" 'boolean t)
envSetVal("mpt" "autoPropagateLock" 'boolean nil)
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

checkHCLOverUnlock

```
mpt checkHCLOverUnlock boolean { t | nil }
```

Description

Specifies whether a hierarchical color lock is placed over an unlocked shape of a different color. The default is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("mpt" "checkHCLOverUnlock")
envSetVal("mpt" "checkHCLOverUnlock" 'boolean t)
envSetVal("mpt" "checkHCLOverUnlock" 'boolean nil)
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

colorCDFCheck

```
mpt colorCDFCheck boolean { t | nil }
```

Description

Specifies whether CDF color is exported. The default is `t`.

GUI Equivalent

Command	*File – Export Color Constraint File*
Field	*Export CDF color* (Export Color Constraint File)

Examples

```
envGetVal("mpt" "colorCDFCheck")
envSetVal("mpt" "colorCDFCheck" 'boolean t)
envSetVal("mpt" "colorCDFCheck" 'boolean nil)
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

Verifying the Consistency of Color Assignments

colorCDFParamPrefix

```
mpt colorCDFParamPrefix string "colorCDFParamPrefix"
```

Description

Specifies the prefix to be used in the Pcell code.

GUI Equivalent

None

Examples

```
envGetVal("mpt" "colorCDFParamPrefix")
envSetVal("mpt" "colorCDFParamPrefix" 'string "cdnColor_")
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

colorConstFileColorAName

```
mpt colorConstFileColorAName string "colorALAyerName"
```

Description

Specifies the name of the color A layer in the Export Color Constraint File form.

GUI Equivalent

Command	*File – Export Color Constraint File*
Field	*Color A Layer Name(s)* (Export Color Constraint File)

Examples

```
envGetVal("mpt" "colorConstFileColorAName")
envSetVal("mpt" "colorConstFileColorAName" 'string "M1_A,M2_A,M3_A")
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

Verifying the Consistency of Color Assignments

colorConstFileColorBName

`mpt colorConstFileColorBName string "`*`colorBLAyerName`*`"`

Description

Specifies the name of the color B layer in the Export Color Constraint File form.

GUI Equivalent

Command	*File – Export Color Constraint File*
Field	*Color B Layer Name(s)* (Export Color Constraint File)

Examples

```
envGetVal("mpt" "colorConstFileColorBName")
envSetVal("mpt" "colorConstFileColorBName" 'string "M1_B,M2_B,M3_B")
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

Verifying the Consistency of Color Assignments

colorConstFileName

`mpt colorConstFileName string "`*`colorConstraintFileName`*`"`

Description

Specifies the name of the log file in the Export Color Constraint File form.

GUI Equivalent

Command	*File – Export Color Constraint File*
Field	*Log file name* (Export Color Constraint File)

Examples

```
envGetVal("mpt" "colorConstFileColorBName")
envSetVal("mpt" "colorConstFileColorBName" 'string "net_color_inpout_file")
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

Verifying the Consistency of Color Assignments

coloredPurposeTypes

```
mpt coloredPurposeTypes string "purpose_names"
```

Description

Specifies the purposes for which shapes on a predefined purpose in this list or whose parent is on a purpose in this list will be colored by the color engine. Coloring will also be applied to shapes on the `drawing` and `pin` purposes, all user-defined purposes, and purposes for which one of these is the parent purpose.

`purpose_names` specifies a list of purposes.

The default value is an empty string (`""`).

If the `explicitColoredPurposes` environment variable is a non-empty string, the `coloredPurposeTypes` environment variable will be ignored.

GUI Equivalent

None

Examples

```
envGetVal("mpt" "coloredPurposeTypes")
envSetVal("mpt" "coloredPurposeTypes" 'string "")
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

Specifying Coloring Purposes

coloringEngineEnabled

```
mpt coloringEngineEnabled boolean { t | nil }
```

Description

Specifies whether to switch on the Virtuoso Multi-Patterning Technology color engine. The default is `nil`, which means switch off the color engine.

GUI Equivalent

Command	*Toolbar – Multiple Patterning*
Field	*Color Engine Switch*

Examples

```
envGetVal("mpt" "coloringEngineEnabled")
envSetVal("mpt" "coloringEngineEnabled" 'boolean t)
envSetVal("mpt" "coloringEngineEnabled" 'boolean nil)
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

Turning the Multiple Patterning Color Engine On and Off

mptActivate

coloringFilterSize

```
mpt coloringFilterSize float float_number
```

Description

Specifies the minimum shape size, in pixels, for which the color of the shape will be displayed. The color of shapes smaller than this size will not be displayed. The default is `5.0`.

GUI Equivalent

Command	*Options – Multiple Patterning*
Field	*Display Color Filter Size* (Multiple Patterning Options)

Examples

```
envGetVal("mpt" "coloringFilterSize")
envSetVal("mpt" "coloringFilterSize" 'float 5.0)
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

colorShiftingLayers

```
mpt colorShiftingLayers string "layerName1 layerName2"
```

Description

Specifies the layers that are color shiftable.

GUI Equivalent

None

Examples

```
envGetVal("mpt" "colorShiftingLayers")
envSetVal("mpt" "colorShiftingLayers" 'string "ndiff")
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

complianceCheckerLimit

```
mpt complianceCheckerLimit integer integer_number
```

Description

Specifies the maximum number of violations that are reported by the methodology compliance checker. When this limit is reached, the errors are not reported and a warning message is displayed in the CIW. The value `0` specifies that there is no limit to the number of violations reported by the methodology compliance checker.

GUI Equivalent

None

Examples

```
envGetVal("mpt" "complianceCheckerLimit")
envSetVal("mpt" "complianceCheckerLimit" 'integer 1000)
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

complianceCheckerReport

```
mpt complianceCheckerReport string "MPT_ComplianceChecker.log"
```

Description

Specifies the file to which you want to save the methodology compliance checker report. The default is `MPTComplianceChecker.log`.

GUI Equivalent

None

Examples

```
envGetVal("mpt" "complianceCheckerReport")
envSetVal("mpt" "complianceCheckerReport" 'string "MPT_ComplianceChecker.log")
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

defaultColoringMethod

```
mpt defaultColoringMethod cyclic { "connectedShapes" | "colorSpacing" }
```

Description

Specifies the method for coloring shapes when the color engine is enabled.

- `connectedShapes`: The color engine automatically colors connected shapes on the same layer.
- `colorSpacing`: The color engine automatically colors connected shapes on the same layer and can change the color of shapes to avoid same-mask spacing violations.

GUI Equivalent

Command	*Options – Multiple Patterning*
Field	*Default Coloring Method* (Multiple Patterning Options)

Examples

```
envGetVal("mpt" "defaultColoringMethod")
envSetVal("mpt" "defaultColoringMethod" 'cyclic "connectedShapes")
envSetVal("mpt" "defaultColoringMethod" 'cyclic "colorSpacing")
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

Using the Multiple Patterning Color Engine

deleteConnectedShapes

```
mpt deleteConnectedShapes boolean { t | nil }
```

Description

Specifies whether the color on the shapes connected to the selected shape must be deleted. The default is `nil`.

GUI Equivalent

Command	*Options – Multiple Patterning*
Field	*Delete color on connected shapes* (Multiple Patterning Options)

Examples

```
envGetVal("mpt" "deleteConnectedShapes")
envSetVal("mpt" "deleteConnectedShapes" 'boolean t)
envSetVal("mpt" "deleteConnectedShapes" 'boolean nil)
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

displayMaskColor

```
mpt displayMaskColor boolean { t | nil }
```

Description

Specifies whether mask coloring will be displayed in the layout. The default is `t`.

GUI Equivalent

Command	*Toolbar – Multiple Patterning*
Field	*Show/Hide Colors*

Examples

```
envGetVal("mpt" "displayMaskColor")
envSetVal("mpt" "displayMaskColor" 'boolean t)
envSetVal("mpt" "displayMaskColor" 'boolean nil)
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

Showing and Hiding Color

displayMaskColorExcludeViaLayers

```
mpt displayMaskColorExcludeViaLayers boolean { t | nil }
```

Description

Specifies that via layers are excluded from the *Show/Hide Color* option on the Multiple Patterning toolbar.

GUI Equivalent

None

Examples

```
envGetVal("mpt" "displayMaskColorExcludeViaLayers")
envSetVal("mpt" "displayMaskColorExcludeViaLayers" 'boolean t)
envSetVal("mpt" "displayMaskColorExcludeViaLayers" 'boolean nil)
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

Showing and Hiding Color

displayMaskColorMode

```
mpt displayMaskColorMode cyclic { "allInOne" | "byMask" | "byLayer" |
     "byLayerAndMask" }
```

Description

Controls the granularity of the *Show/Hide Color* function on the Multiple Patterning toolbar.

- `allInOne`: Lets you enable or disable the visibility of all mask colors of all LPPS.
- `byMask`: Lets you enable or disable a specific mask color. For example, `mask1Color` of all colorable LPPs is set to visible or invisible.
- `byLayer`: Lets you enable or disable all mask colors for a specific layer. For example, `mask1Color`, `mask2Color`, and `mask3Color` for `Metal1` layer is set to visible or invisible.
- `byLayerAndMask`: Lets you enable or disable a mask color for a specific layer. For example, `Metal1- Mask1Color` is set to visible or invisible.

The default is `"allInOne"`.

GUI Equivalent

None

Examples

```
envGetVal("mpt" "displayMaskColorMode")
envSetVal("mpt" "displayMaskColorMode" 'cyclic "allInOne")
envSetVal("mpt" "displayMaskColorMode" 'cyclic "byMask")
envSetVal("mpt" "displayMaskColorMode" 'cyclic "byLayer")
envSetVal("mpt" "displayMaskColorMode" 'cyclic "byLayerAndMask")
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

displaySystemColor

```
mpt displaySystemColor boolean { t | nil }
```

Description

Specifies whether unlocked colors will be displayed in the layout. The default is `t`.

GUI Equivalent

None

Examples

```
envGetVal("mpt" "displaySystemColor")
envSetVal("mpt" "displaySystemColor" 'boolean t)
envSetVal("mpt" "displaySystemColor" 'boolean nil)
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

dontColorPCells

```
mpt dontColorPCells boolean { t | nil }
```

Description

Specifies whether the recoloring of Pcells will be prevented when running *ReColor All* or *Update Color* Using the Multiple Patterning Toolbar or using the mptReColor or mptUpdateColor SKILL function. The default is `nil`.

GUI Equivalent

Command	*Options – Multiple Patterning*
Field	*Don't color Pcells* (Multiple Patterning Options)

Examples

```
envGetVal("mpt" "dontColorPCells")
envSetVal("mpt" "dontColorPCells" 'boolean t)
envSetVal("mpt" "dontColorPCells" 'boolean nil)
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

drawSurroundingOn

```
mpt drawSurroundingOn boolean { t | nil }
```

Description

Specifies whether the the color information is displayed at the top-level with the `EIP` suffix appended at the edit in place level.

GUI Equivalent

None

Examples

```
envGetVal("mpt" "drawSurroundingOn")
envSetVal("mpt" "drawSurroundingOn" 'boolean t)
envSetVal("mpt" "drawSurroundingOn" 'boolean nil)
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

enableHCLCreation

```
mpt enableHCLCreation cyclic { "all" | "level1" | "level1_pins" | "none" }
```

Description

Controls the creation of hierarchical color locks on cells.

- `all`: HCL can be created on all cell shapes from any level.
- `level1`: HCL can be created on all cell shapes at the instance level (`level1`) only.
- `level1_pins`: HCL can be created on cell pins at instance level (`level1`) only.
- `none`: HCL cannot be created on cell shapes.

The default is `all`.

GUI Equivalent

None

Examples

```
envGetVal("mpt" "enableHCLCreation")
envSetVal("mpt" "enableHCLCreation" 'cyclic "all")
envSetVal("mpt" "enableHCLCreation" 'cyclic "level1")
envSetVal("mpt" "enableHCLCreation" 'cyclic "level1_pins")
envSetVal("mpt" "enableHCLCreation" 'cyclic "none")
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

Post-Select Locking and Unlocking

enableHCLCreationOnPcells

```
mpt enableHCLCreationOnPcells cyclic { "all" | "level1" | "level1_pins" | "none" }
```

Description

Controls the creation of hierarchical color locks on Pcells.

- `all`: HCL can be created on all Pcell shapes from any level.
- `level1`: HCL can be created on all Pcell shapes at the instance level (`level1`) only.
- `level1_pins`: HCL can be created on Pcell pins at instance level (`level1`) only.
- `none`: HCL cannot be created on Pcell shapes.

The default is `all`.

GUI Equivalent

None

Examples

```
envGetVal("mpt" "enableHCLCreationOnPcells")
envSetVal("mpt" "enableHCLCreationOnPcells" 'cyclic "all")
envSetVal("mpt" "enableHCLCreationOnPcells" 'cyclic "level1")
envSetVal("mpt" "enableHCLCreationOnPcells" 'cyclic "level1_pins")
envSetVal("mpt" "enableHCLCreationOnPcells" 'cyclic "none")
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

enableMarkersToMaskColors

```
mpt enableMarkersToMaskColors boolean { t | nil }
```

Description

Specifies whether the *Markers to Mask* icon is visible in the Multiple Patterning toolbar. The default is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("mpt" "enableMarkersToMaskColors")
envSetVal("mpt" "enableMarkersToMaskColors" 'boolean t)
envSetVal("mpt" "enableMarkersToMaskColors" 'boolean nil)
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

Converting Markers to Mask Colors

`mptMarkersToMaskColors`

enforceWSPColor

```
mpt enforceWSPColor boolean { t | nil }
```

Description

Specifies that WSP color is prioritized in case of a color conflict. The default is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("mpt" "enforceWSPColor")
envSetVal("mpt" "enforceWSPColor" 'boolean t)
envSetVal("mpt" "enforceWSPColor" 'boolean nil)
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

explicitColoredPurposes

`mpt explicitColoredPurposes string "`*`purpose_names`*`"`

Description

Specifies the purposes for which only the shapes on a predefined purpose in this list will be colored by the color engine.

`purpose_names` specifies a list of purposes.

The default value is `""`, coloring will be applied only to shapes on the `drawing` and `pin` purposes, all user-defined purposes, and purposes for which one of these is the parent purpose. A shape on any other predefined purpose will not be colored unless its purpose is specified by the `coloredPurposeTypes` environment variable.

GUI Equivalent

None

Examples

```
envGetVal("mpt" "explicitColoredPurposes")
envSetVal("mpt" "explicitColoredPurposes" 'string "")
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

Specifying Coloring Purposes

extractorStopLevel

```
mpt extractorStopLevel integer integer_number
```

Description

Specifies the stop level for the shapes to be considered for coloring.

For example,

- `0`: top level to top level only
- `1`: top level to top level, and top level to level 1 shapes
- `2`: all of the above plus level 1 to level 1, and level 1 to level 2 shapes

The default value is `1`.

GUI Equivalent

Command	*Options – Multiple Patterning*
Field	*Hierarchy Stop Level* (Multiple Patterning Options)

Examples

```
envGetVal("mpt" "extractorStopLevel")
envSetVal("mpt" "extractorStopLevel" 'integer 1)
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

forcePcellRecolorOnEval

```
mpt forcePcellRecolorOnEval boolean { t | nil }
```

Description

Specifies that the Pcell is recolored based on evaluation independent of the state of the coloring engine. This enables use model based on the on-demand coloring feature when the coloring engine is disabled. The default is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("mpt" "forcePcellRecolorOnEval")
envSetVal("mpt" "forcePcellRecolorOnEval" 'boolean t)
envSetVal("mpt" "forcePcellRecolorOnEval" 'boolean nil)
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

forceRandomOnLayerType

```
mpt forceRandomOnLayerType cyclic { "none" | "cut" | "metal" | "all"}
```

Description

Forces the unclustered shape color or default shape assignment mode to be set to random. You can specify this environment variable when you want to use a random color assignment even if default colors are assigned for specific layers. When you specify this environment variable, the default colors are not deleted.

- `none`: If a default color is set, the random mode has no priority on the color assignment over the default color for the specific layers.
- `cut`: The random mode is forced only on cut layers.
- `metal`: The random mode is forced for the metal layers only. The cut layers will have the default behavior.
- `all`: The random mode is forced on all layers, cut and metal.

The default is `none`.

GUI Equivalent

None

Examples

```
envGetVal("mpt" "forceRandomOnLayerType")
envSetVal("mpt" "forceRandomOnLayerType" 'cyclic "none")
envSetVal("mpt" "forceRandomOnLayerType" 'cyclic "cut")
envSetVal("mpt" "forceRandomOnLayerType" 'cyclic "metal")
envSetVal("mpt" "forceRandomOnLayerType" 'cyclic "all")
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

globalColorShiftingPolicy

```
mpt globalColorShiftingPolicy cyclic { "cluster" | "layer" | "none" }
```

Description

Controls the color shift types that the coloring engine can use to resolve color conflicts on instances.

- `cluster`: Cluster shifts are used. Cluster references are created.

  Note: Clusters are groups of shapes that should color shift together. The shapes can be connected or they can be within same mask spacing.
- `layer`: Only layer shifts are used. Cluster references are not created.
- `none`: Cells are not shifted. Cluster references are not created.

The default is `none`.

GUI Equivalent

None

Examples

```
envGetVal("mpt" "globalColorShiftingPolicy")
envSetVal("mpt" "globalColorShiftingPolicy" 'cyclic "cluster")
envSetVal("mpt" "globalColorShiftingPolicy" 'cyclic "layer")
envSetVal("mpt" "globalColorShiftingPolicy" 'cyclic "none")
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

globalColorShiftingPolicyForPcells

```
mpt globalColorShiftingPolicyForPcells cyclic { "cluster" | "layer" | "none" }
```

Description

Controls the color shift types that the coloring engine can use to resolve color conflicts on Pcells.

- `cluster`: Cluster shifts are used. Cluster references are created.
- `layer`: Only layer shifts are used. Cluster references are not created.
- `none`: Pcells are not shifted. Cluster references are not created.

The default is `cluster`.

GUI Equivalent

None

Examples

```
envGetVal("mpt" "globalColorShiftingPolicy")
envSetVal("mpt" "globalColorShiftingPolicy" 'cyclic "cluster")
envSetVal("mpt" "globalColorShiftingPolicy" 'cyclic "layer")
envSetVal("mpt" "globalColorShiftingPolicy" 'cyclic "none")
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

hideStitchingTools

```
mpt hideStitchingTools boolean { t | nil }
```

Description

Specifies whether the *Stitch* and *UnStitch* buttons are hidden in the Multiple Patterning toolbar. The default is `t`.

GUI Equivalent

None

Examples

```
envGetVal("mpt" "hideStitchingTools")
envSetVal("mpt" "hideStitchingTools" 'boolean t)
envSetVal("mpt" "hideStitchingTools" 'boolean nil)
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

Using Stitch and UnStitch

Using Stitch for Multiple Patterning Violations

lockAllHCLPolicy

```
mpt lockAllHCLPolicy cyclic { "noHCL" | "cell" | "pcell" | "cellAndPcell" }
```

Description

Controls the lock all behavior of hierarchical color locks.

- `noHCL`: No HCL is created.
- `cell`: HCL is created on the cells as defined by the cell HCL policy, `enableHCLCreation`. No HCL is created on Pcell shapes.
- `pcell`: HCL is created on the Pcells as defined by the Pcell HCL policy, `enableHCLCreationOnPcells`. No HCL is dropped on cells.
- `cellAndPcell`: HCL is created on the cells and Pcells as defined by the HCL policies on cell and Pcells.

The default is `noHCL`.

Note: This environment variable does not influence the instance shapes locked by the `propagateLocksToConnectedShapes` or `autoPropagateLock` environment variables.

GUI Equivalent

None

Examples

```
envGetVal("mpt" "lockAllHCLPolicy")
envSetVal("mpt" "lockAllHCLPolicy" 'cyclic "noHCL")
envSetVal("mpt" "lockAllHCLPolicy" 'cyclic "cell")
envSetVal("mpt" "lockAllHCLPolicy" 'cyclic "pcell")
envSetVal("mpt" "lockAllHCLPolicy" 'cyclic "cellAndPcell")
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

mergeColoredPacket

```
mpt mergeColoredPacket boolean { t | nil }
```

Description

Specifies whether colored shapes are displayed using the merged colored and uncolored display packets. The default is `t`. When set to `nil`, colored shapes are displayed using only the colored display packet.

GUI Equivalent

None

Examples

```
envGetVal("mpt" "mergeColoredPacket")
envSetVal("mpt" "mergeColoredPacket" 'boolean t)
envSetVal("mpt" "mergeColoredPacket" 'boolean nil)
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

Displaying Colored Shapes

mptConstraintGroup

```
mpt mptConstraintGroup string "mpt_ConstraintGroup"
```

Description

Specifies the name of the MPT constraint group to be used. The default value is `""`. You can explicitly specify the name of the MPT constraint group.

GUI Equivalent

None

Examples

```
envGetVal("mpt" "mptConstraintGroup")
envSetVal("mpt" "mptConstraintGroup" 'string "")
envSetVal("mpt" "mptConstraintGroup" 'string "virtuosoMPTSetup")
envSetVal("mpt" "mptConstraintGroup" 'string "virtuosoMPTSetup_custom")
envSetVal("mpt" "mptConstraintGroup" 'string "virtuosoMPTSetup_override")
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

Specifying Coloring Purposes

onlyCheckActiveWSP

```
mpt onlyCheckActiveWSP boolean { t | nil }
```

Description

Specifies whether only the shapes that overlap an active colored width spacing pattern (WSP) track are colored, based on their position relative to the track and the setting of the trackColoringOnlySnappedShapes environment variable. The default is `nil`; the shapes overlapping any colored WSP track are colored based on the same criteria.

GUI Equivalent

None

Examples

```
envGetVal("mpt" "onlyCheckActiveWSP")
envSetVal("mpt" "onlyCheckActiveWSP" 'boolean t)
envSetVal("mpt" "onlyCheckActiveWSP" 'boolean nil)
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

Coloring Shapes that Overlap Colored Tracks

overrideLockOnConnectedShapes

```
mpt overrideLockOnConnectedShapes boolean { t | nil }
```

Description

Specifies whether the locked color on a shape can change when there is a color conflict with a connected shape. The default is `nil`; a locked color shape cannot change color to avoid a color conflict with a connected shape.

GUI Equivalent

Command	*Options – Multiple Patterning*
Field	*Override lock on connected shapes* (Multiple Patterning Options)

Examples

```
envGetVal("mpt" "overrideLockOnConnectedShapes")
envSetVal("mpt" "overrideLockOnConnectedShapes" 'boolean t)
envSetVal("mpt" "overrideLockOnConnectedShapes" 'boolean nil)
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

propagateLocksToConnectedShapes

```
mpt propagateLocksToConnectedShapes boolean { t | nil }
```

Description

Specifies whether color locking operates on the selected shapes and the shapes connected to them on the same layer when a lock is initiated from the Multiple Patterning toolbar and the color engine is on. The default is `nil`; color locking applies only to the selected shapes.

GUI Equivalent

Command	*Options – Multiple Patterning*
Field	*Propagate locks to connected shapes* (Multiple Patterning Options)

Examples

```
envGetVal("mpt" "propagateLocksToConnectedShapes")
envSetVal("mpt" "propagateLocksToConnectedShapes" 'boolean t)
envSetVal("mpt" "propagateLocksToConnectedShapes" 'boolean nil)
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

propagateAnySameMaskState

```
mpt propagateAnySameMaskState boolean { t | nil }
```

Description

Specifies whether locks are propagated to all the shapes on the same routing layer for a net class when the Constraint Manager *Same Mask* pre-coloring constraint is `true` and the *Mask Name* pre-coloring constraint is `any` for the net class. The default is `nil`; locks are not propagated when the *Same Mask* pre-coloring constraint is `true` and the *Mask Name* pre-coloring constraint is `any`.

GUI Equivalent

None

Examples

```
envGetVal("mpt" "propagateAnySameMaskState")
envSetVal("mpt" "propagateAnySameMaskState" 'boolean t)
envSetVal("mpt" "propagateAnySameMaskState" 'boolean nil)
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

pvsDeckFile

`mpt pvsDeckFile string` *`any_file_name`*

Description

Specifies the name of the rule deck file used by Pegasus. The default value is `""`.

GUI Equivalent

None

Examples

```
envGetVal("mpt" "pvsDeckFile")
envSetVal("mpt" "pvsDeckFile" 'string "ruleFile.pvl")
```

Related Topics

List of Virtuoso MPT Environment Variables

pvsLayerMapFile

`mpt pvsLayerMapFile string` *`any_file_name`*

Description

Specifies the name of the layer map file used by Pegasus. The default value is `""`.

GUI Equivalent

None

Examples

```
envGetVal("mpt" "pvsLayerMapFile")
envSetVal("mpt" "pvsLayerMapFile" 'string "layerMap.layermap")
```

Related Topics

List of Virtuoso MPT Environment Variables

reColorGUIScope

```
mpt reColorGUIScope cyclic { "currentCellView" | "outdatedCells" | "all"}
```

Description

Sets and queries the *ReColor All* GUI mode.

- `currentCellView`: Sets the GUI to *Current CellView* mode.
- `outdatedCells`: Sets the GUI to *Outdated Cells* mode.
- `all`: Sets the GUI to *all* mode.

GUI Equivalent

Command	*Toolbar – Multiple Patterning*
Field	*Recolor Selected – ReColor All – Recolor*

Examples

```
envGetVal("mpt" "reColorGUIScope")
envSetVal("mpt" "reColorGUIScope" 'cyclic "currentCellView")
envSetVal("mpt" "reColorGUIScope" 'cyclic "outdatedCells")
envSetVal("mpt" "reColorGUIScope" 'cyclic "all")
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

Using the Multiple Patterning Toolbar

reColorReadOnlyCellView

```
mpt reColorReadOnlyCellView boolean { t | nil }
```

Description

Specifies whether read-only cellviews can be colored by `mptReColor` and the Multiple Patterning toolbar *ReColor All* function. The default is `nil`.

GUI Equivalent

Command	*Options – Multiple Patterning*
Field	*Recolor readOnly cellviews* (Multiple Patterning Options)

Examples

```
envGetVal("mpt" "reColorReadOnlyCellView")
envSetVal("mpt" "reColorReadOnlyCellView" 'boolean t)
envSetVal("mpt" "reColorReadOnlyCellView" 'boolean nil)
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

Recolor All

shiftCutColorFromToolbarButton

```
mpt shiftCutColorFromToolbarButton boolean { t | nil }
```

Description

Shifts the cut color of vias when the *Shift Colors* icon is selected on the MPT toolbar. The default is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("mpt" "shiftCutColorFromToolbarButton")
envSetVal("mpt" "shiftCutColorFromToolbarButton" 'boolean t)
envSetVal("mpt" "shiftCutColorFromToolbarButton" 'boolean nil)
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

showCDFchecks

```
mpt showCDFchecks boolean { t | nil }
```

Description

Displays the *CDF Color Check* and *Net Color Constraint Check* options in the *Verify* menu in VLS XL, and *Check* menu in VSE XL. It also displays the *Net Color File* option on the CDL Out form and the *Export Color Constraint File* in VSE XL. The default is `nil`.

GUI Equivalent

None

Examples

```
envGetVal("mpt" "showCDFchecks")
envSetVal("mpt" "showCDFchecks" 'boolean t)
envSetVal("mpt" "showCDFchecks" 'boolean nil)
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

trackColoringOnlySnappedShapes

```
mpt trackColoringOnlySnappedShapes boolean { t | nil }
```

Description

Specifies whether read-only cellviews can be colored by `mptReColor` and the Multiple Patterning toolbar *Recolor All* function. The default is `t`.

Specifies whether shapes inherit the color of the track only when snapped to the track. When set to `t`, color inheritance depends on the track type and the position of the shape relative to the track:

- For track patterns, the color of the track is inherited only if the centerline of the shape is aligned with the centerline of the track.
- For WSP tracks, the color of the track is inherited only if the centerline of the wire or pathSeg and its edges align with the track's centerline and edges. For polygons and rectangles, color inheritance requires that the bounding box of the shape align with the track's edges and that the minimal length for the bounding box in the x or y direction equal the width of the track.

When set to `nil`, all the shapes overlapping a colored track inherit the color of the track, regardless of the alignment. The default is `t`.

GUI Equivalent

None

Examples

```
envGetVal("mpt" "trackColoringOnlySnappedShapes")
envSetVal("mpt" "trackColoringOnlySnappedShapes" 'boolean t)
envSetVal("mpt" "trackColoringOnlySnappedShapes" 'boolean nil)
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

Coloring Shapes that Overlap Colored Tracks

unclusteredShapeColor

```
mpt unclusteredShapeColor cyclic { "gray" | "random" | "asIs" | "layerDefault" }
```

Description

Sets the color assignment for shapes on multiple patterning layers that do not belong to a cluster when the MPT color engine is activated.

- `gray`: No color assignment.
- `random`: Shapes are randomly assigned to an available mask color for the layer.
- `asIs`: Shapes with a color assignment will be unchanged unless a color violation exists, then the color engine can change the color. Shapes without a color assignment but that are same-mask spacing or less from another same-layer shape will be colored using the layer default color, if set, or will be randomly colored; otherwise, gray shapes will remain gray.
- `layerDefault`: Shapes are assigned to the default color for the layer.

The default is `"asIs"`.

GUI Equivalent

Command	*Options – Multiple Patterning*
Field	*Default Shape Assignment* (Multiple Patterning Options)

Examples

```
envGetVal("mpt" "unclusteredShapeColor")
envSetVal("mpt" "unclusteredShapeColor" 'cyclic "gray")
envSetVal("mpt" "unclusteredShapeColor" 'cyclic "random")
envSetVal("mpt" "unclusteredShapeColor" 'cyclic "asIs")
envSetVal("mpt" "unclusteredShapeColor" 'cyclic "layerDefault")
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

updateColorOnActivate

```
mpt updateColorOnActivate cyclic { "ask" | "always" | "never" }
```

Description

Determines whether coloring will be updated when the color engine is turned on and the design contains outdated coloring data.

- `ask`: The *Automatic UpdateColor* dialog box appears.
- `always`: Hierarchical coloring is automatically performed on designs containing outdated coloring data.
- `never`: Coloring is not updated.

The default is `never`.

GUI Equivalent

None

Examples

```
envGetVal("mpt" "updateColorOnActivate")
envSetVal("mpt" "updateColorOnActivate" 'cyclic "ask")
envSetVal("mpt" "updateColorOnActivate" 'cyclic "always")
envSetVal("mpt" "updateColorOnActivate" 'cyclic "never")
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

When the Color Engine Is Activated

copyMPAttributes

```
cdba copyMPAttributes boolean { t | nil }
```

Description

Specifies whether the coloring information from source objects is copied as-is to the corresponding destination objects when copying or using MakeCell. The default is `nil`.

GUI Equivalent

Command	*Options – Multiple Patterning*
Field	*Maintain color while copying* (Multiple Patterning Options)

Examples

```
envGetVal("cdba" "copyMPAttributes")
envSetVal("cdba" "copyMPAttributes" 'boolean t)
envSetVal("cdba" "copyMPAttributes" 'boolean nil)
```

Related Topics

List of Virtuoso MPT Environment Variables

Environment Variable Functions

B

MPT Functions

This chapter provides an overview of SKILL functions used with Virtuoso® Multi-Patterning Technology (MPT). Some functions support the multiple patterning color engine and others support Virtuoso database access and technology data for Multi-Patterning Technology (MPT).

- Virtuoso MPT SKILL Functions
- Virtuoso SKILL Functions for MPT

Virtuoso MPT SKILL Functions

The SKILL functions in the following tables support the multiple patterning color engine. These functions are described in detail in *MPT SKILL Functions* in *Virtuoso Layout Suite SKILL Reference*.

Table B-1 Coloring Method

SKILL Function	Description
`mptActivate( g_activate ) => t / nil`	Turns the color engine on or off.
`mptGetDefaultColoringMethod( ) => t_coloringMethod`	Returns the session default coloring method.
`mptGetLayerColoringMethod( d_cellViewID [ t_layerName ] ) => t_coloringMethod / t_SDLcolorMethod / nil`	Returns either the coloring method for the given layer, or the coloring method for the session, the design, and layers with a specified coloring method.
`mptSetDefaultColoringMethod( t_coloringMethod ) => t / nil`	Sets the session default coloring method.
`mptSetLayerColoringMethod( { d_cellViewID | d_techID } l_layers t_coloringMethod ) => t / nil`	Sets the coloring method for the technology database, the design, or specified layers in the design.

Table B-2 Color Assignment

SKILL Function	Description
`mptGetLayerDefaultColor( d_techFileID [ t_layerName ] ) => t_color / l_layerColor / nil`	Returns either the default color for the given layer, or a list of layer-default color pairs.

Table B-2 Color Assignment

SKILL Function	Description
`mptGetLockDefaultColors( ) => t / nil`	Indicates whether the layer default color specified by `mptSetLayerDefaultColor` will be locked when assigned to a shape.
`mptSetLayerDefaultColor(` *`t_layer`* *`t_color`* `) => t / nil`	Specifies the color assignment for newly created shapes on the given layer.
`mptSetLockDefaultColors( ) => t / nil`	Specifies whether the layer default color specified by `mptSetLayerDefaultColor` will be locked when assigned to a shape.

Table B-3 Hierarchical Coloring

SKILL Function	Description
`mptPropagateLocks( [ d_cellviewID [ g_convertGroups ]] ) => t / nil`	Propagates all the locks visible from the top level to connected shapes through the hierarchy.
`mptPropagateSameMaskGroups( d_cellViewID ) => t / nil`	Propagates same mask groups in the given cellview through the hierarchy.

Table B-4 Color Locking

SKILL Function	Description
`mptLockAll( d_cellviewID [ l_layerNames ] ) => t / nil`	Locks all the top-level shapes of the specified cellview with their current color. You can optionally lock only the shapes on specific colorable layers. Gray color shapes are not locked.
`mptUnlockAll( d_cellViewID [ l_layerNames ] ) => t / nil`	Unlocks all the colored shapes that are locked at the top level of the specified cellview. You can optionally unlock only the shapes on specific colorable layers. Top-level shapes with color attribute locks (dbLocks) are unlocked with their current color. Hierarchical color locks at the current editing hierarchy level are removed.

Table B-5 Batch Coloring

SKILL Function	Description
`mptCleanClusters( d_cellViewID ) => t / nil`	Cleans out-of-date cluster information in the cellview which can help to reduce the design size and/or improve performance.

Table B-5 Batch Coloring

SKILL Function	Description
`mptDeleteClusters( d_cellViewID [ l_layers [ g_depth ]] ) => t / nil`	Removes coloring clusters in the specified design.
`mptGetOutdatedDesigns( d_cellViewID [ t_mode ] ) => l_outdated / nil`	Returns a list of the cellviews in the design hierarchy for which the color information is outdated.
`mptGetUpToDateDesigns( d_cellViewID ) => l_upToDate / nil`	Returns a list of the cellviews in the design hierarchy for which the color information is up to date.
`mptReColor( d_cellviewID [ l_layers [ g_depth ]] ) => t / nil`	Recolors the entire cellview. **Note:** By default, this function will color the entire design, even though coloring for read-only cellviews cannot be saved. This is useful for determining if the design is colorable. To prevent recoloring of read-only cells of the top design, set the `reColorReadOnlyCellView` environment variable to `nil`. `envSetVal("mpt" "reColorReadOnlyCellView" 'boolean nil)`
`mptReColorFromShapes( l_shapes ) => t / nil`	Recolors the shapes in the list and the shapes that are connected to them. If the coloring method is `managed`, the color of the shapes that are within same-mask spacing of the shapes in the list will be updated according to the clustering algorithm.
`mptUpdateColor( d_cellViewID [ l_layers [ g_depth ]] ) => t / nil`	Updates color information. Similar to `mptReColor` but runs faster for cases when only part of the design is out of date, because valid coloring information is not recomputed.

Table B-6 Stitch and Unstitch

SKILL Function	Description

`mptHiStitch(`
`)`
`=> nil`
	Turns on Stitch mode.

`mptHiUnStitch(`
`)`
`=> nil`
	Turns on Unstitch mode.

Table B-7 Data Input/Output

SKILL Function	Description

`mptLPPMergeToColor(`
`d_cellviewID`
`g_libName`
`g_viewName`
`l_mapping`
`[` *`g_colorState`* `]`
`)`
`=> t / nil`
	Converts the layout into Virtuoso multi-pattern conformance format for designs that use two or more separate layer-purpose pairs to represent multi-patterning mask information.

`mptMarkersToColoredBlockges(`
`l_purposes`
`d_cellViewID`
`)`
`=> t / nil`
	Transforms the shapes on the specified purposes of colored layers in the given cellview to colored blockages on the same layer.

`mptMarkersToMaskColors(`
`l_purposes`
`g_drawingPurpose`
`d_cellViewID`
`g_locked`
`)`
`=> t / nil`
	Transforms the shapes in the given cellview to the drawing purpose and assigns color and color state to the shape from the color specified by the color marker purpose.

Table B-8 Color Display

SKILL Function	Description

`mptGetColoredPurposes(`
`    d_cellViewID`
`    )`
`    => l_lpp / nil`
	Returns a list of the colored layer-purpose pairs for the cellview.

`mptIsMaskColorShown(`
`    t_layerName`
`    x_maskNumber`
`    )`
`    => t / nil`
	Checks whether the specified mask color is visible for the layer.

`mptShowMaskColor(`
`    l_layers`
`    x_maskNumber`
`    g_toShow`
`    )`
`    => t / nil`
	Shows or hides the color for the specified mask on the given layers.

Table B-9 Multiple Patterning Toolbar Functions

SKILL Function	Description

`mptDoToolbarAction(`
`    g_toolbarFunc`
`    )`
`    => t / nil`
	Runs the specified Multiple Patterning toolbar function.

Virtuoso SKILL Functions for MPT

This section lists the SKILL functions associated with the following:

- Virtuoso Design Environment SKILL Functions for MPT
- Virtuoso Technology Database SKILL Functions for MPT
- Virtuoso Layout Suite L Palette Assistant SKILL Functions for MPT

Table B-10 lists the SKILL functions that operate on database objects and are grouped by function.

Table B-10 Virtuoso Design Environment SKILL Functions for MPT

Category	Function
Shape-based Coloring	`dbGetShapeColor`
	`dbIsShapeColored`
	`dbIsShapeColoringAllowed`
	`dbIsShapeColorLocked`
	`dbRemoveLayerColorData`
	`dbSetShapeColor`
	`dbSetShapeColorLocked`
Blockage Coloring	`dbGetBlockageColor`
	`dbIsBlockageColored`
	`dbIsBlockageColoringAllowed`
	`dbSetBlockageColor`

Table B-10 Virtuoso Design Environment SKILL Functions for MPT

Category	Function
Via Coloring	`dbGetViaCutLayerControl`
	`dbGetViaLayer`
	`dbGetViaLayer1Control`
	`dbGetViaLayer2Control`
	`dbGetViaLayerNumColorMasks`
	`dbIsViaColorStateLayerLocked`
	`dbIsViaColorStateLocked`
	`dbSetViaColorStateLayerLocked`
	`dbSetViaColorStateLocked`
	`dbSetViaCutLayerControl`
	`dbSetViaLayer1Control`
	`dbSetViaLayer2Control`
Coloring of track patterns	`dbGetTrackPatternFirstTrackColor`
	`dbIsTrackPatternColorAlternating`
	`dbIsTrackPatternColored`
	`dbIsTrackPatternColoringAllowed`
	`dbSetTrackPatternColorAlternating`
	`dbSetTrackPatternFirstTrackColor`
Relationship Coloring	`dbAddObjectToGroup`
	`dbCreateDiffMaskGroup`
	`dbCreateSameMaskGroup`
	`dbDeleteObjectFromGroup`
	`dbGetSameMaskDiffMaskGroups`
	`dbGetShapeSameMaskGroups`

Table B-10 Virtuoso Design Environment SKILL Functions for MPT

Category	Function
Hierarchical Shape Color Query	`dbColorShapeQuery`
	`dbColorShapeQuery2`
	`dbGetColoredOccShapes`
	`dbGetShapeEffectiveColor`
	`dbSetOccShapeColor`
	`dbSetOccShapeColorLocked`
Color shifting	`dbGetColorModel`
	`dbGetIntegrationColorModel`
	`dbSetColorModel`
	`dbSetIntegrationColorModel`
Color shifting controls	`dbCellViewAreLayerShiftsValid`
	`dbCellViewClearLayerShifts`
	`dbCellViewHasLayerShifts`
	`dbCellViewInitForLayerShifting`
	`dbCellViewUpdateLayerShifts`
	`dbInstClearLayerShifts`
	`dbInstGetLayerShifts2`
	`dbInstHasLayerShifts`
	`dbInstSetLayerShifts2`

For detailed information on the SKILL functions in Table B-10, refer to *Multi-Patterning Technology Functions* in *Virtuoso Design Environment SKILL Reference*.

Table B-11 lists the MPT SKILL functions that operate on technology databases and are grouped by function.

Table B-11 Virtuoso Technology Database SKILL Functions for MPT

Category	Function
Mask color	`techGetLayerNumColorMasks`
	`techSetLayerNumColorMasks`

Table B-11 Virtuoso Technology Database SKILL Functions for MPT

Category	Function
Coloring of vias	`techGetStdViaDefCutColoring`
	`techGetTechCutColoring`
	`techSetStdViaDefCutColoring`
	`techSetTechCutColoring`
	`techIsStdViaDefCutColoringSet`
Color shifting	`techGetIntegrationColorModel`
	`techSetIntegrationColorModel`

For detailed information on the SKILL functions in Table B-11, refer to *MPT Functions* in *Virtuoso Technology Data SKILL Reference*.

Table B-12 lists MPT SKILL functions associated with the Virtuoso Layout Suite L Palette assistant.

Table B-12 Virtuoso Layout Suite L Palette Assistant SKILL Functions for MPT

Category	Function
MPT Support	`pteCloseMPTSupportMode`
	`pteMPTSupportMode`
	`pteSetActiveLppColor`
	`pteSetActiveLppColorLock`
	`pteSetShowLockedColors`
	`pteSetShowUnlockedColors`
Layers Panel	`pteShowMPTLPP`
	`pteToggleShowMPTLPP`

For detailed information on the SKILL functions in Table B-12, refer to *Palette Assistant Functions* in *Virtuoso Layout Suite SKILL Reference*.

C

MPT Forms Reference

This appendix describes the Virtuoso Multi-Patterning Technology (MPT) forms:

- Export Color Constraint File
- Multiple Patterning Options
- Markers To Mask Colors

Note: Many of the options described in this section have a corresponding environment variable noted next to the description. For more information, see List of Virtuoso MPT Environment Variables.

Export Color Constraint File

Use the **Export Color Constraint File** form to generate the color constraint file to perform Layout Versus Schematic (LVS) coloring checks.

For details on using the Export Color Constraint File GUI, see LVS Coloring Check.

Color A Layer Name(s) specifies the name of the color A layer.
Environment variable: colorConstFileColorAName

Color A Layer Name(s) specifies the name of the color B layer.
Environment variable: colorConstFileColorBName

Export CDF color specifies that the CDF color must be exported.
Environment variable: colorCDFCheck

Log file name specifies the name of the log file.
Environment variable: colorConstFileName

Multiple Patterning Options

Use the **Multiple Patterning Options** form to change the default multiple patterning environment variable settings.

For details on using the Multiple Patterning options GUI, see Using the Multiple Patterning Options Form.

Default Coloring Method specifies the coloring method when the color engine is enabled. Choices are *Connected shapes only (interactive)* and *Connected shapes & color spacing* (*managed)*.
Environment variable: defaultColoringMethod

Default Shape Assignment sets the color assignment for shapes on multiple patterning layers that do not belong to a cluster when the color engine is activated. Choices are `gray`, `random`, `asIs`, and `layerDefault`.
Environment variable: unclusteredShapeColor

Maintain color while copying specifies that the coloring information from source objects will be copied as-is to the corresponding destination objects when copying or using MakeCell.
Environment variable: copyMPAttributes

Propagate locks to connected shapes automatically propagates locks to connected shapes when the lock is initiated from the Multiple Patterning toolbar.
Environment variable: propagateLocksToConnectedShapes

Advanced Options

Don't color Pcells prevents the recoloring of Pcells when running *ReColor All* or *Update Color* from the Multiple Patterning toolbar or using the `mptReColor` or `mptUpdateColor` SKILL function.
Environment variable: dontColorPCells

Recolor readOnly cellviews recolors read-only cellviews when running *ReColor All* or *Update Color* from the Multiple Patterning toolbar or using the `mptReColor` or `mptUpdateColor` SKILL function. If not selected, only editable cellviews are recolored by those SKILL and GUI functions.
Environment variable: reColorReadOnlyCellView

Allow shifting of locked shapes allows color shifting of color-locked shapes. By default, shapes must be unlocked before shifting colors.
Environment variable: allowLockShiftOverride

Override lock on connected shapes allows color-locked shapes to change color to avoid color conflicts with connected shapes when a lock is initiated from the Multiple

Patterning toolbar. By default, color-locked shapes cannot change color to avoid color conflicts.
Environment variable: overrideLockOnConnectedShapes

Propagate locks while editing automatically propagates locks to connected shapes on the same layer when the lock is initiated by editing (for example, by adding, moving, or stretching a shape, or assigning a locked color using Property Editor)
Environment variable: autoPropagateLock

Note: The *Propagate Locks While Editing* and *Override Lock on Connected Shapes* options are enabled only if you select the *Propagate Locks to Connected Shapes* option.

Delete color on connected shapes deletes the color on the shapes connected to the selected shape.
Environment variable: deleteConnectedShapes

Display Color Filter Size specifies the minimum shape size, in pixels, for which the color of the shape will be displayed. The color of shapes smaller than this size will not be displayed.
Environment variable: coloringFilterSize

Hierarchy Stop Level specifies the level to which shapes will be considered when coloring. A value of 1 considers only top-level and level-1 shapes. A value of 0 considers only shapes on the current level.
Environment variable: extractorStopLevel

Markers To Mask Colors

Use the Markers To Mask Colors form to convert the imported stream data that uses duplicate shapes to represent coloring. One shape is transformed to the `drawing` purpose and its color is assigned based on the purpose of the duplicate shape.

For details on using the Markers to Mask Colors GUI, see Converting Markers to Mask Colors.

Drawing Purpose specifies that shapes on this purpose will be transformed to the `drawing` purpose.

Mask1 Marker Purpose specifies that marker shapes on this purpose assign mask1Color to the duplicate shape on the `drawing` purpose.

Mask2 Marker Purpose specifies that marker shapes on this purpose assign mask2Color to the duplicate shape on the `drawing` purpose.

Mask3 Marker Purpose specifies that marker shapes on this purpose assign mask3Color to the duplicate shape on the `drawing` purpose. This purpose applies only for layers with more than two masks.

Mask4 Marker Purpose specifies that marker shapes on this purpose assign mask4Color to the duplicate shape on the `drawing` purpose. This purpose applies only for layers with more than two masks.

Lock Shapes locks the color state on shapes.

Remove Marker Shapes removes the marker purspose shapes.

Run Recolor All recolors all cellviews.

Save Cellviews automatically saves the modified cellviews in the hierarchy.