

Virtuoso Fluid Guard Ring Developer Guide

**Product Version ICADVM20.1
October 2020**

© 2016-2020 Cadence Design Systems, Inc. All rights reserved.
Printed in the United States of America.

Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

Open SystemC, Open SystemC Initiative, OSCI, SystemC, and SystemC Initiative are trademarks or registered trademarks of Open SystemC Initiative, Inc. in the United States and other countries and are used with permission.

Trademarks: Trademarks and service marks of Cadence Design Systems, Inc. contained in this document are attributed to Cadence with the appropriate symbol. For queries regarding Cadence's trademarks, contact the corporate legal department at the address shown above or call 800.862.4522. All other trademarks are the property of their respective holders.

Restricted Permission: This publication is protected by copyright law and international treaties and contains trade secrets and proprietary information owned by Cadence. Unauthorized reproduction or distribution of this publication, or any portion of it, may result in civil and criminal penalties. Except as specified in this permission statement, this publication may not be copied, reproduced, modified, published, uploaded, posted, transmitted, or distributed in any way, without prior written permission from Cadence. Unless otherwise agreed to by Cadence in writing, this statement grants Cadence customers permission to print one (1) hard copy of this publication subject to the following conditions:

1. The publication may be used only in accordance with a written agreement between Cadence and its customer.
2. The publication may not be modified in any way.
3. Any authorized copy of the publication or portion thereof must include all original copyright, trademark, and other proprietary notices and this permission statement.
4. The information contained in this document cannot be used in the development of like products or software, whether for internal or external use, and shall not be used for the benefit of any other party, whether or not for consideration.

Disclaimer: Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information.

Restricted Rights: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor

Contents

<u>Preface</u>	7
<u>Scope</u>	8
<u>Licensing Requirements</u>	8
<u>Related Documentation</u>	8
<u>What's New and KPNS</u>	8
<u>Installation, Environment, and Infrastructure</u>	8
<u>Technology Information</u>	9
<u>Virtuoso Tools</u>	9
<u>SKILL Documents</u>	9
<u>Relative Object Design and Inherited Connections</u>	10
<u>Application Notes</u>	10
<u>Additional Learning Resources</u>	11
<u>Video Library</u>	11
<u>Virtuoso Videos Book</u>	11
<u>Rapid Adoption Kits</u>	11
<u>Help and Support Facilities</u>	12
<u>Customer Support</u>	12
<u>Feedback about Documentation</u>	13
<u>Typographic and Syntax Conventions</u>	14
1	
<u>Introduction to Fluid Guard Rings</u>	17
<u>Fluid Shapes: Concepts</u>	17
<u>Fluid Pcells</u>	19
<u>Fluid Guard Rings: Concepts</u>	20
<u>VLS-based Fluid Guard Rings</u>	21
<u>Custom Fluid Guard Rings</u>	25

2

Fluid Guard Ring Infrastructure 27

<u>Class Hierarchy in VFO Infrastructure</u>	27
<u>Implementation Class</u>	27
<u>Filling Class</u>	35
<u>Enclosure Class</u>	38
<u>Protocol Class</u>	39

3

Pitch Support in Fluid Guard Ring 43

<u>Overview of Pitch and Grid</u>	43
<u>Calculating the Fluid Shape Data using Pitch Parameters</u>	44
<u>Calculating the Centerline of Path Style Fluid Shape using Pitch Parameters</u>	45
<u>Calculating the Width of Path Style Fluid Shape using Pitch Parameters</u>	49

4

Develop and Define a Fluid Guard Ring Device 51

<u>Extending the VFO Infrastructure</u>	51
<u>Extending the Implementation Class</u>	51
<u>Extending the Filling Class</u>	52
<u>Defining Fluid Guard Ring Devices</u>	53
<u>Defining the Device Class</u>	53
<u>Declaring the Device</u>	56
<u>Specifying Device Properties</u>	58
<u>Adding or Modifying CDF Parameters</u>	59

5

Customize Create Guard Ring Form 63

<u>Modifying the Existing Create Guard Ring Form</u>	63
<u>Adding New User-Defined GUI Components to the Form</u>	63
<u>Updating the Existing GUI Components on the Form</u>	64
<u>Updating Properties of User-Defined GUI Components</u>	65
<u>Example: Hiding Existing Fields and Adding New Check Box to the Form</u>	66

Virtuoso Fluid Guard Ring Developer Guide

<u>Pitch Parameter Support in the Create Guard Ring Form</u>	67
<u>Creating a New Create Guard Ring Form</u>	68
<u>Using the Create Guard Ring Form</u>	71
<u>Other SKILL Functions for Create Form Modifications</u>	73
<u>Triggers</u>	73
<u>Queues</u>	73
<u>Field Prompts in vfoGRSetCreateFormFieldProp SKILL Function</u>	74

6

<u>Write Customized Fluid Editing Commands</u>	77
<u>Defining Fluid Editing Commands</u>	78
<u>Pitch Handling Support for Editing Commands</u>	79
<u>Stretch Command</u>	79
<u>Merge Command</u>	79

7

<u>Methodology to Maintain Versions of Implementation Class</u>	83
---	----

8

<u>Fluid Guard Ring Packaging in PDK</u>	89
<u>Procedure for Initializing Customized FGR Devices</u>	90

A

<u>Best Practices for Developing a Fluid Guard Ring</u>	93
---	----

B

<u>Fluid Guard Ring Environment Variables</u>	97
<u>Setting Environment Variables</u>	97
<u>.cdsenv File</u>	97
<u>.cdsinit File</u>	98
<u>CIW</u>	98
<u>Displaying the Current Value of an Environment Variable</u>	98

Virtuoso Fluid Guard Ring Developer Guide

<u>List of Environment Variables</u>	99
<u>fgrPostEditPitchCorrection</u>	100
<u>fgrWrapPlaceAtMinimumDistance</u>	101
<u>fluidGuardRingInstallPath</u>	102
<u>grEnclosedBy</u>	103
<u>grMode</u>	104
<u>keepGuardRingEndsConnected</u>	105
<u>vfoShowOnlyFluidShapeForDrag</u>	106

C

<u>Fluid Guard Rings Known Problems and Solutions</u>	107
---	-----

D

<u>Performance Improvement in Tunnel Command</u>	113
--	-----

Preface

The Virtuoso Layout Suite L (Layout L) provides an innovative infrastructure with capabilities to create and implement layout designs. One of the important offerings of Layout L includes a menu-driven programmable feature for installing, creating, and editing fluid guard ring (FGR) devices, which are a type of fluid Pcells.

This developer guide covers information about the Virtuoso Fluid Object (VFO) infrastructure on which FGR creation and edit capabilities are based. It describes how this infrastructure can be extended to build on the existing strengths and customize it as per process and design requirements.

This user guide is aimed at developers and designers of integrated circuits who want to harness the usability and productivity benefits of FGR devices in Layout L. It assumes that you are familiar with:

- Virtuoso design environment and application infrastructure mechanisms designed to support consistent operations between all Cadence® tools
- Applications used to design and develop integrated circuits in the Virtuoso design environment, notably Virtuoso Layout Suite and Virtuoso Schematic Editor
- Design and use of parameterized cells
- OpenAccess version 2.2 technology file
- Component description format (CDF)

Virtuoso allows you to work with customized FGRs developed using the VFO infrastructure that adheres to the added or modified user-defined capabilities or features. In other words, the customized FGRs are based on capabilities that are not shipped as part of Virtuoso. This guide focuses on assisting PDK developers in developing customized FGRs by extending the VFO infrastructure.

This preface contains the following topics:

- [Scope](#)
- [Licensing Requirements](#)
- [Related Documentation](#)
- [Additional Learning Resources](#)

- [Customer Support](#)
- [Feedback about Documentation](#)
- [Typographic and Syntax Conventions](#)

Scope

Unless otherwise noted, the functionality described in this guide can be used in both mature node (for example, IC6.1.8) and advanced node and methodologies (for example, ICADVM20.1) releases.

Label	Meaning
(ICADVM20.1 Only)	Features supported only in ICADVM20.1 advanced nodes and advanced methodologies releases.
(IC6.1.8 Only)	Features supported only in mature node releases.

Licensing Requirements

For using the FGR capabilities, you need to have a licensed Layout L installation.

For information about licensing in the Virtuoso design environment, see [Virtuoso Software Licensing and Configuration Guide](#).

Related Documentation

What's New and KPNS

- [Virtuoso Fluid Guard Ring What's New](#).
- [Virtuoso Fluid Guard Ring Known Problems and Solutions](#)

Installation, Environment, and Infrastructure

- [Cadence Installation Guide](#)

Virtuoso Fluid Guard Ring Developer Guide

Preface

- [*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 SKILL Reference*](#)

Virtuoso Tools

- [*Virtuoso Layout Suite SKILL Reference*](#)
- [*Virtuoso Layout Suite XL User Guide*](#)
- [*Virtuoso Schematic Editor L User Guide*](#)
- [*Virtuoso Space-based Router User Guide*](#)
- [*Virtuoso Design Rule Driven Editing User Guide*](#)
- [*Virtuoso Relative Object Design User Guide*](#)
- [*Virtuoso Parameterized Cell Reference*](#)
- [*Design Data Translator's Reference*](#)

SKILL Documents

- The SKILL programming language is documented in the following manuals:
 - [*Virtuoso Design Environment SKILL Reference*](#)
 - [*Cadence SKILL Language User Guide*](#)
 - [*Cadence SKILL Language Reference*](#)
 - [*Cadence SKILL Development Reference*](#)
 - [*Cadence SKILL IDE User Guide*](#)
- SKILL access to other applications is provided in the following manuals:

- ❑ [*Virtuoso Technology Data SKILL Reference*](#)
- ❑ [*Virtuoso Layout Suite SKILL Reference*](#)
- ❑ [*Virtuoso Schematic Editor SKILL Reference*](#)
- ❑ [*Cadence User Interface SKILL Reference*](#)
- ❑ [*Cadence Interprocess Communication SKILL Reference*](#)

Relative Object Design and Inherited Connections

- ❑ [*Virtuoso Relative Object Design User Guide*](#)
- ❑ [*Virtuoso Schematic Editor L User Guide*](#)

Application Notes

The following FGR-specific application notes that are available on the [Cadence Online Support](#) website provide some useful additional information:

- [Customizing Create Guard Ring Form](#)

This document explains how the Create Guard Ring form can be customized using specific triggers and SKILL APIs.

- [Adding and Managing CDF Parameters for Fluid Guard Rings](#)

The document shows how to add and update the CDF parameters and attributes that affect the geometry of a fluid guard ring instance.

- [Adding User-Defined Capabilities to Fluid Guard Rings](#)

This document shows how to add user-defined capabilities or features to FGRs that are currently not supported using the supplied Install Guard Ring form.

- [Creating Fluid SKILL Pcells](#)

This document explains the features of fluid SKILL Pcells.

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 [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 on Virtuoso fluid guard ring functionality and related Virtuoso tools:

- [Virtuoso Layout Design Basics](#)
- [Virtuoso Layout Pro: T1 Environment and Basic Commands \(L\)](#)
- [Virtuoso Layout Pro: T2 Create and Edit Commands \(L\)](#)
- [Virtuoso Layout Pro: T3 Basic Commands \(XL\)](#)
- [Virtuoso Connectivity-Driven Layout Transition](#)
- [Virtuoso Layout for Advanced Nodes \(ICADVM20.1 Only\)](#)

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.

Typographic and Syntax Conventions

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

<i>text</i>	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.
<i>z_argument</i>	Indicates text that you must replace with an appropriate argument value. The prefix (in this example, <i>z_</i>) 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 <i>key argument</i> . 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.

Virtuoso Fluid Guard Ring Developer Guide

Preface

Virtuoso Fluid Guard Ring Developer Guide

Preface

Introduction to Fluid Guard Rings

To implement complex designs, developers mostly use parameterized cells (Pcells) that are created using Cadence® SKILL language. For detailed information on pcells, see the *Virtuoso Parameterized Cell SKILL Reference*.

Though pcells provide a strong effective programmatic way to edit the designs, with the growing complexities, there was a need for an intuitive capability to visually modify shapes in the design. This need brought into being the concept of fluid shapes and the fluid guard rings (FGR), which are discussed in this chapter. An FGR is a fluid SKILL pcell with powerful creation and graphical editing capabilities. A fluid SKILL pcell is a pcell with one or more fluid shape.

Fluid Shapes: Concepts

Any shape that is registered using the `dbSetFluidShape` SKILL API, as shown below, is a fluid shape.

```
dbSetFluidShape( d_shapeId t_shapeName ) => t / nil
```

Note: It is recommended that you create a fluid shape on a non-maskable layer purpose pair (LPP).

A fluid shape is selectable from the top level in Virtuoso Layout Suite L (Layout L). Therefore, it supports editing features like any other level 0 shape.

You can use the following SKILL APIs to retrieve the fluid shape information:

```
dbGetFluidShapes( d_cvId ) => list(fluidShape1 fluidShape2 ...) / nil
dbGetFluidShapeByName( d_cvId t_shapeName ) => d_fluidShapeId / nil
dbGetFluidShapeName( d_fluidShapeId ) => t_shapeName
dbIsFluidShape( d_shapeId ) => t / nil
```

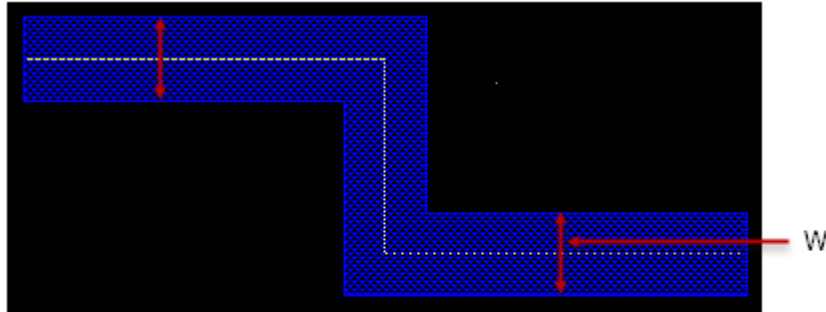
A fluid shape is defined using the following two parameters:

- `shapeType`
- `shapeData`

Virtuoso Fluid Guard Ring Developer Guide

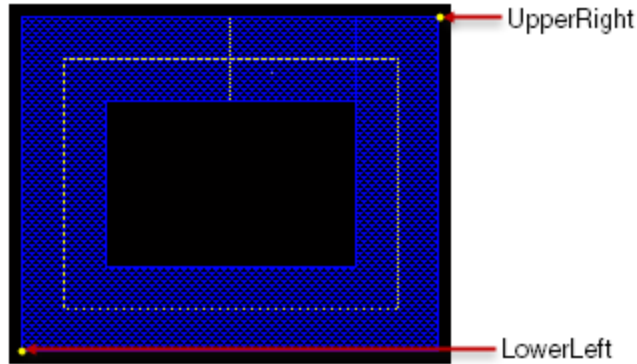
Introduction to Fluid Guard Rings

The infrastructure supports three styles of fluid shapes: path, polygon, and rectangle. The following images illustrate different styles of fluid shapes and their representation in the infrastructure:



Path Style

```
width = W
centerline = (x1,y1) (x2,y2) ... (xn,yn)
shapeType = "path"
shapeData = (W((x1,y1) (x2,y2) ... (xn,yn)))
```

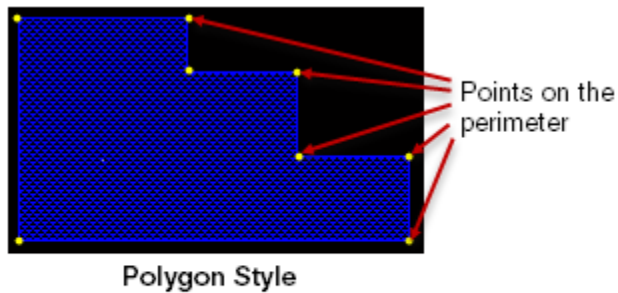


Rectangle Style

```
bBox = (x1,y1) (x2,y2) = (LL) (UR)
shapeType = "rect"
shapeData = ((x1,y1) (x2,y2))
```

Virtuoso Fluid Guard Ring Developer Guide

Introduction to Fluid Guard Rings



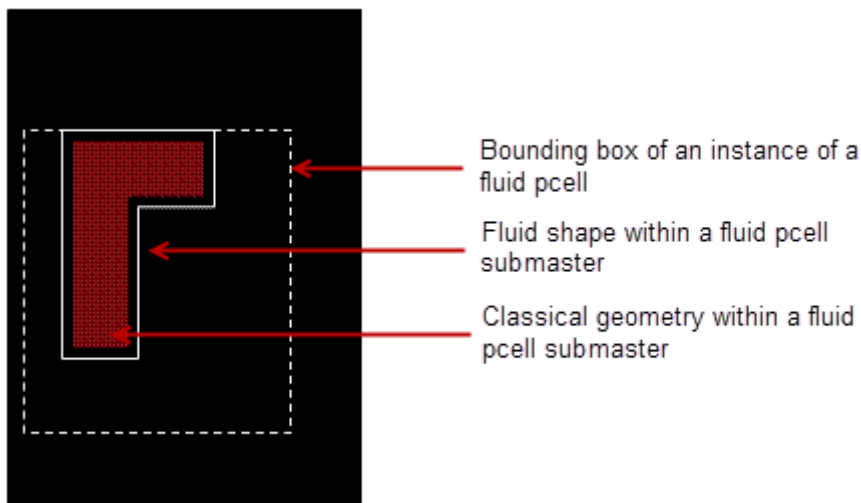
```
perimeter = (x1,y1) (x2,y2) ... (xn,yn)
shapeType = "poly"
shapeData = ((x1,y1) (x2,y2) ... (xn,yn))
```

When you edit a fluid shape, a set of SKILL updater functions associated to it are called from the associated pcell submaster.

Fluid Pcells

A pcell with one or more fluid shapes make a fluid pcell. Fluid pcell can be edited graphically (like shapes) and whose behavior in response to editing commands can be defined and customized using SKILL language. Some examples of fluid pcells are: guard rings, space filling capacitors, and filling structures.

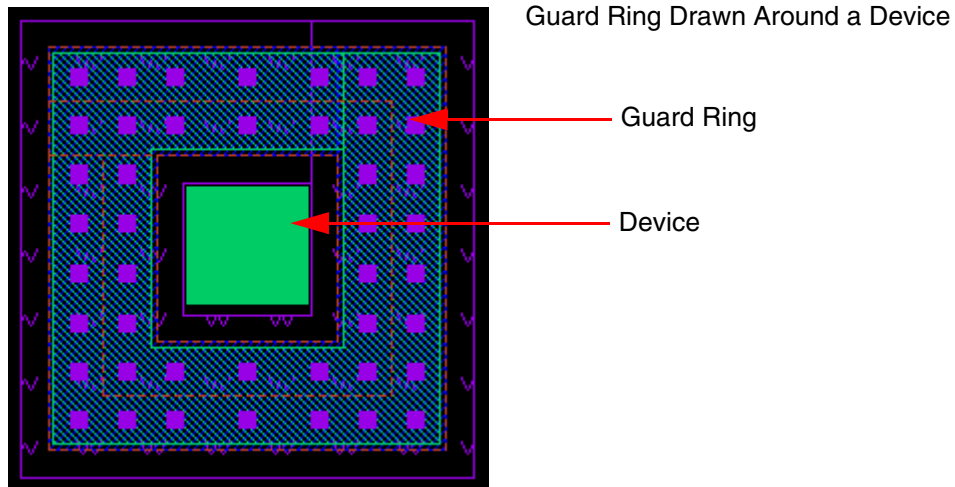
The following figure shows a representation of a fluid pcell:



For detailed information about fluid pcells, refer to the [*Creating Fluid SKILL Pcells*](#) application note available on the [Cadence Online Support](#) website.

Fluid Guard Rings: Concepts

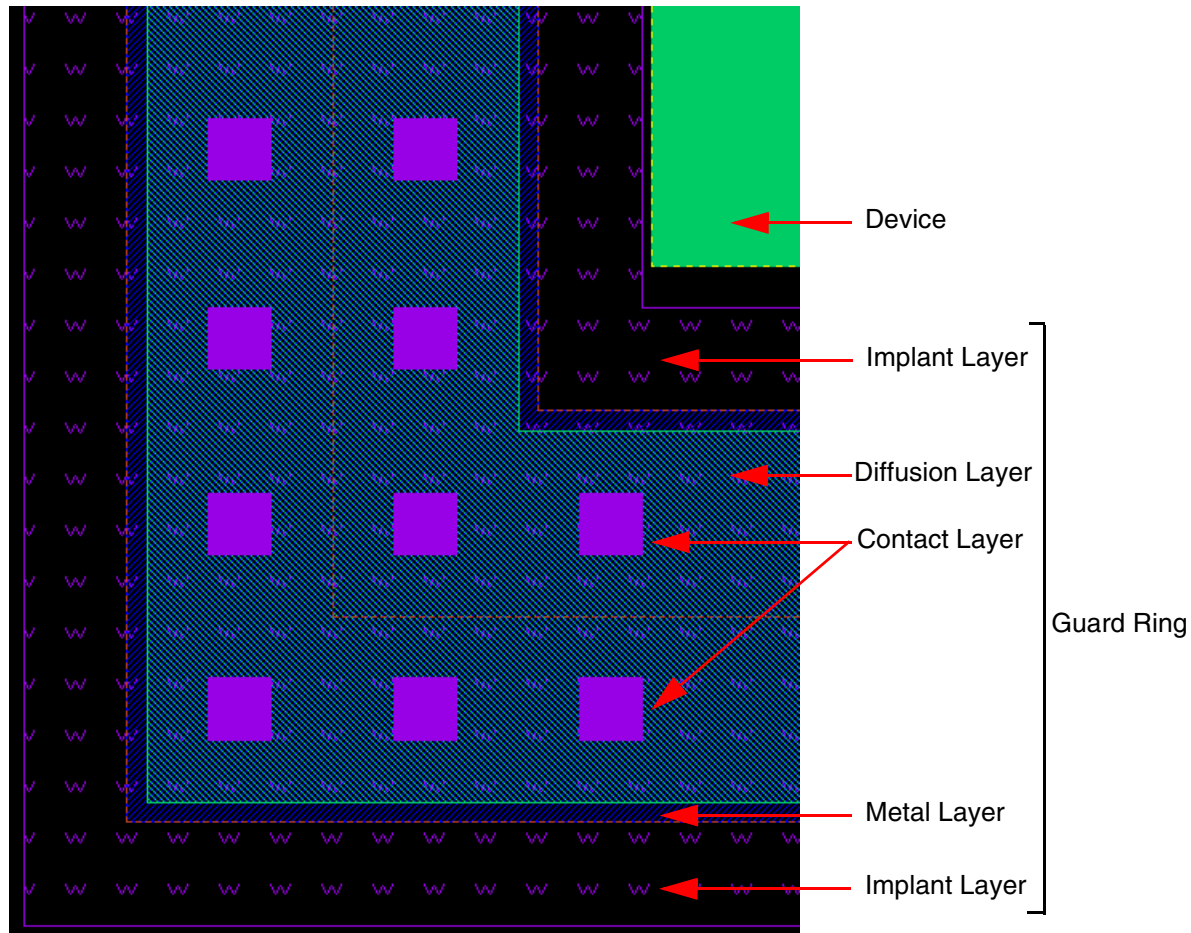
FGRs are a type of fluid pcells in which all shapes are drawn based on the fluid shape points. Unlike standard pcells, FGRs can be created and graphically edited on the canvas itself.



Virtuoso Fluid Guard Ring Developer Guide

Introduction to Fluid Guard Rings

The following figure shows a cross-section of an FGR device.



Virtuoso provides support for the following two types of FGRs:

- VLS-based Fluid Guard Rings
- Custom **Fluid Guard Rings**

VLS-based Fluid Guard Rings

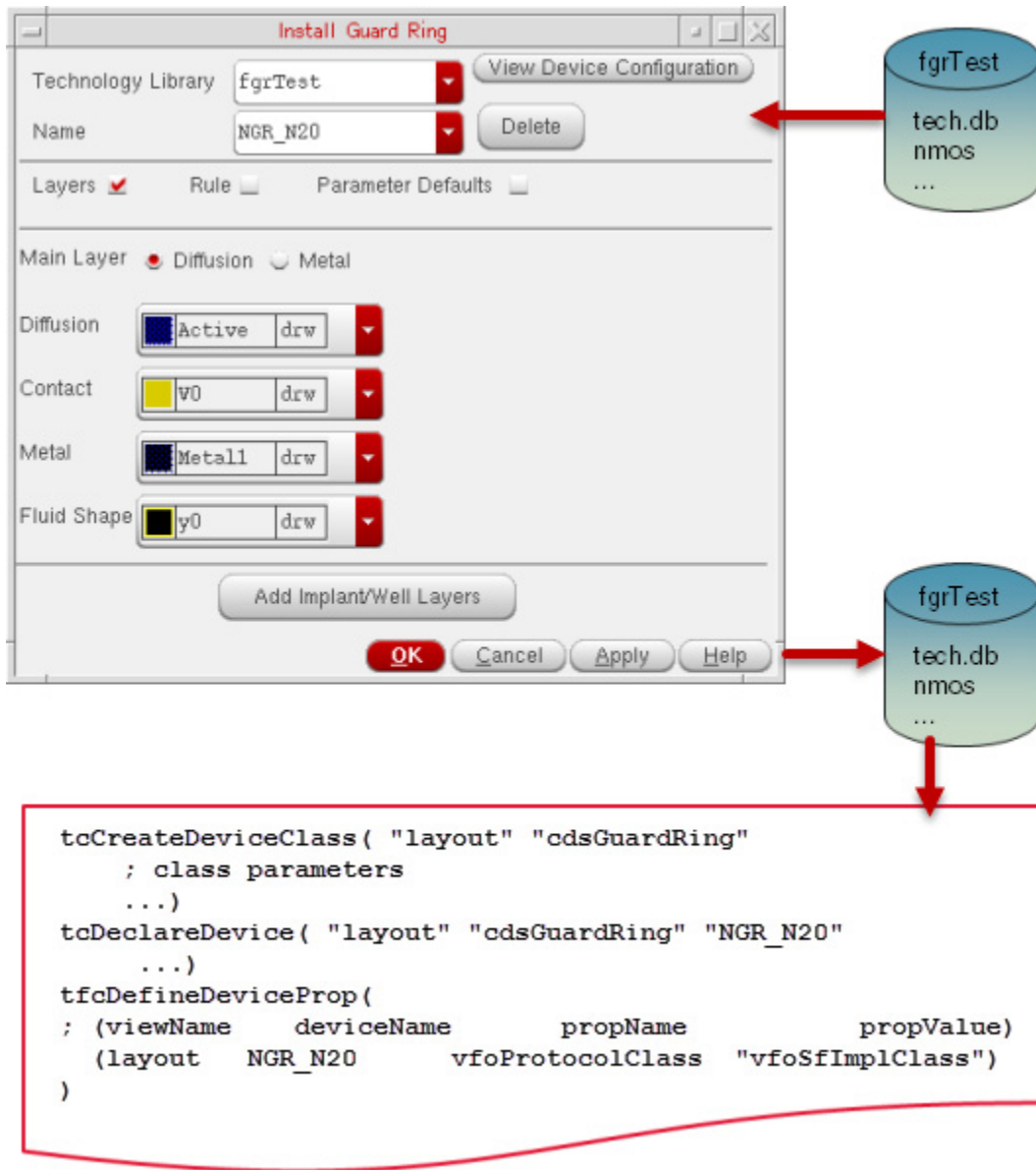
The VLS-based FGRs are provided by default with Virtuoso that automatically loads the related set of implementation files (`vfo*.ils`) at the time of initialization. These files provide the Virtuoso Fluid Object (VFO) infrastructure that enables you to install, create, and edit the VLS-based FGRs.

The VLS-based solution primarily involves installation, creation, and editing of an FGR using the default forms that can be accessed from Layout L. For example:

Virtuoso Fluid Guard Ring Developer Guide

Introduction to Fluid Guard Rings

- Install an FGR using the Install Guard Ring form that allows you to define and save an FGR to the required technology library.



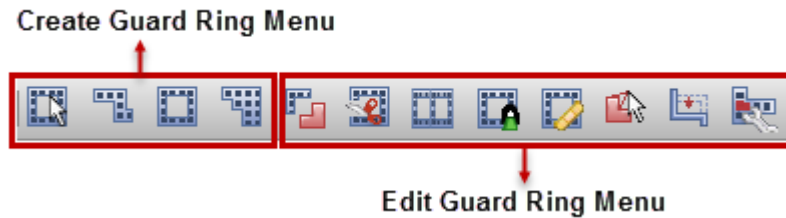
Once an FGR device gets installed through this FGR installation form, the `tcCreateDeviceClass`, `tcDeclareDevice`, and `tfcDefineDeviceProp` constructs in the technology file get created or updated accordingly.

- Create an FGR using the Create Guard Ring form displayed using the *Create – Fluid Guard Ring* menu. This form allows you to create an FGR of the required shape. To access the related toolbar, through the *Window – Toolbars* menu, click the *Guardring*

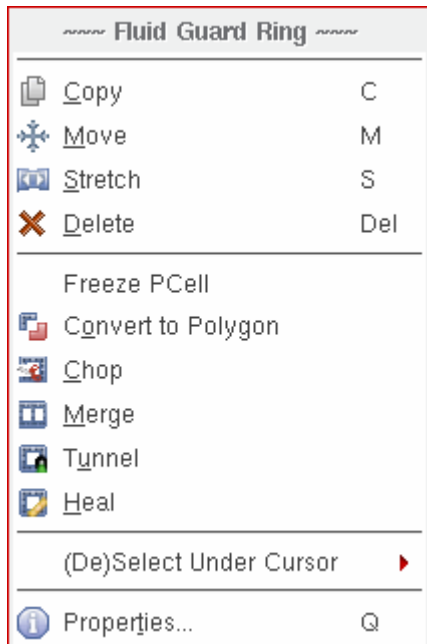
Virtuoso Fluid Guard Ring Developer Guide

Introduction to Fluid Guard Rings

submenu. The following FGR toolbar is displayed and it provides the icons that enable you to create and edit FGR instances.



- Edit the fluid shape in an FGR using one of the following methods:
 - ❑ Use the edit commands available in the FGR toolbar shown above.
 - ❑ Use the additional editing commands that are available in the context-sensitive *Fluid Guard Ring* menu, which is displayed when you right-click any FGR instance. Alternatively, you can access these commands from the *Edit* menu of the Layout L window.



Virtuoso Fluid Guard Ring Developer Guide

Introduction to Fluid Guard Rings

- ❑ Use the Edit Instance Properties form displayed after you select a fluid shape and do a right click it. This form allows you to change the parameters defined in the VFO infrastructure.

Edit Instance Properties

< > Apply 1/1 Common

Attribute Connectivity **Parameter** Property ROD

Show Which Params? ☒ all ☐ shape ☐ style ☐ dimensions ☐ debug

Formal Version: 1

Main Layer: Diffusion

Contact Spacing Method: distribute

Contact Placement Method: fill45-path-poly

Shape Data: (0.6 0.6) (1.0 0.6) (1.0 0.3) (1.4 0.3) (1.4 0.0) (0.0 0.0)

Perimeter: (0.6 0.6) (1.0 0.6) (1.0 0.3) (1.4 0.3) (1.4 0.0) (0.0 0.0)

Contact Width: 0.001

Contact Space: 0.056

Match Contact Enclosures: ☐

Minimum Metal over Contact (W): 0.007

Minimum Metal over Contact (L): 0.03

Minimum Diffusion over Contact (W): 0.007

Minimum Diffusion over Contact (L): 0.03

☐ Display CDF Parameter Name

Convert To Mosaic OK Cancel Apply Help

For detailed usage information about the VLS-based FGRs, refer to [Virtuoso Fluid Guard Ring User Guide](#) and [Virtuoso Fluid Guard Ring Frequently Asked Questions](#).

Custom Fluid Guard Rings

A customized FGR can either be a pcell developed in SKILL with customer-specific capabilities and features, or a hierarchical FGR pcell developed by encapsulating a customer-developed pcell guard ring.

Virtuoso allows you to develop custom FGRs by extending the classes in VFO infrastructure to incorporate user-defined capabilities or features. VFO infrastructure is also enhanced to provide pitch support in custom FGRs. For detailed information about the VFO infrastructure and pitch support, refer to [Chapter 2, “Fluid Guard Ring Infrastructure”](#) and [Chapter 3, “Pitch Support in Fluid Guard Ring”](#).

For information about how to extend classes in infrastructure to develop custom FGRs, refer to [Chapter 4, “Develop and Define a Fluid Guard Ring Device”](#).

The customization of FGR can include customization of Create Guard Ring form, which enables you to add user-specific fields to the form and customization of fluid editing commands. For detailed information, refer to [Chapter 5, “Customize Create Guard Ring Form.”](#) and [Chapter 6, “Write Customized Fluid Editing Commands.”](#)

You can maintain two versions of an implementation class that is available in the VFO infrastructure to contain different drawing codes. For detailed information, refer to [Chapter 7, “Methodology to Maintain Versions of Implementation Class.”](#)

These custom FGRs can also be read in third-party tools by loading relevant VFO infrastructure files. For detailed information about how to manually load the VFO infrastructure, refer to [Chapter 8, “Fluid Guard Ring Packaging in PDK.”](#)

While developing a custom FGR, Cadence recommends some best practices to achieve reliable results. For detailed information, refer to [Appendix A, “Best Practices for Developing a Fluid Guard Ring.”](#)

There are some FGR-specific Layout L environment variables that can be used to control the behavior of FGRs in Layout Editor environment. For detailed information, refer to [Appendix B, “Fluid Guard Ring Environment Variables.”](#)

Virtuoso Fluid Guard Ring Developer Guide

Introduction to Fluid Guard Rings

Fluid Guard Ring Infrastructure

The implementation of the fluid guard ring (FGR) solution resides in a SKILL object-oriented infrastructure known as Virtuoso Fluid Object (VFO) infrastructure. This consists of classes and their methods. This chapter covers information about these classes. You can extend these classes to control the behavior of FGRs in your layout designs.

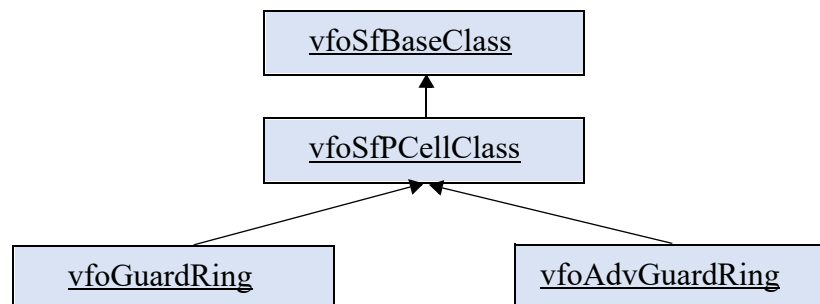
Class Hierarchy in VFO Infrastructure

The VFO infrastructure is primarily based on the following SKILL++ classes:

- Implementation Class
- Filling Class
- Enclosure Class
- Protocol Class

Implementation Class

The implementation class structure has the following four classes:



The `vfoGuardRing` and `vfoAdvGuardRing` are the two main implementation classes that contain the `draw()` method. If you are implementing fluid guard ring with pitch support, use the `vfoAdvGuardRing` class.

Virtuoso Fluid Guard Ring Developer Guide

Fluid Guard Ring Infrastructure

vfoSfBaseClass

- Slots: *None*
- Methods: *None*

vfoSfPCellClass

- Slots
- Methods

Slots

The `vfoSfPCellClass` class has the following slots:

Slots for fluid shape parameters	
<code>modelLpp</code>	The layer-purpose pair (LPP) for fluid shape. <i>Data type:</i> list
<code>shapeType</code>	The type of fluid shape, that is, path, rectangle, or polygon.
<code>shapeData</code>	The shape points.
Slots for editing parameters	
<code>vfoProtocolClass</code>	The editing class. <i>Data type:</i> string
<code>keepOuts</code>	The regions where tunnels are being created in the following format: <code>(list(layer bbox))</code> <i>Data type:</i> list
Slots for layers	
<code>mainLpp</code>	The master LPP mapped to fluid shape, normally a metal or diffusion layer.
<code>highlightLpp</code>	The annotate LPP.
<code>tmpLpp</code>	The temporary layer to draw intermediate shapes on. <i>Data type:</i> list
Slots for connectivity	
<code>termName</code>	The name of the terminal to which the fluid shape must be connected.

Virtuoso Fluid Guard Ring Developer Guide

Fluid Guard Ring Infrastructure

pinName	The name of the pin to which the fluid shape must be connected.
Slots for enclosure	
enclosureClass	The vfoSfEnclosureClass class that is used for drawing the enclosing layers.
enclosures	The layers that should enclose the main LPP, normally, implant and well layers.
Slots for debugging	
debug	A positive integer value.
hide_keepouts	A Boolean value, that is, t or nil.
do_something	A Boolean value, that is, t or nil.
Slots for contact filling class	
fillClass	The name of the contact filling class.
decompositionMode	The contact filling mode.
Slots for initialization of formal and CDF parameters	
Note: <i>Do not update</i> the following slots during customization.	
cv	
cdf	
superMaster	
shapeObj	

Methods

The vfoSfPCellClass class has the following methods:

- vfoSupportsVersionCache ((pcell vfoSfPCellClass))
- vfoSfFilling ((pcell vfoSfPCellClass))
- vfoSfRegisterFluidShape ((pcell vfoSfPCellClass) d_obj)
- vfoSfInitialize ((pcell vfoSfPCellClass))
- vfoSfFinalize ((pcell vfoSfPCellClass))

Virtuoso Fluid Guard Ring Developer Guide

Fluid Guard Ring Infrastructure

- `vfoSfCreateEnclosure ((pcell vfoSfPCellClass)
 @key (className 'vfoSfEnclosureClass)
 layerop lpp (coverInterior nil)
 @rest initargs)`
- `vfoSfCreateFilling ((pcell vfoSfPCellClass)
 @key (className 'vfoSfFillSafe)
 layerop cutLpp lpp grid cw cs_x cs_y
 overlap_x overlap_y
 @rest initargs)`
- `vfoSfAddFeedback ((pcell vfoSfPCellClass) propName value)`
- `vfoSfDraw ((pcell vfoSfPCellClass))`
- `vfoSfShapesPerLpp ((pcell vfoSfPCellClass) lpp)`
- `vfoSfCreatePins ((pcell vfoSfPCellClass))`
- `vfoSfDrawEnclosures ((pcell vfoSfPCellClass) shape)`
- `vfoSfCutKeepOuts ((pcell vfoSfPCellClass))`
- `vfoGRGeometry ((pcell vfoSfPCellClass))`

vfoGuardRing

- Slots
- Methods: *None*

Slots

The `vfoGuardRing` class has the following slots:

<code>techFile</code>	The name of the technology files. <i>Data type:</i> string
<code>vfoGRImpl</code>	The name of the implementation class. <i>Data type:</i> string
<code>classVersion</code>	The version of the VFO source code. <i>Data type:</i> integer
<code>formalVersion</code>	This slot is used only for internal processing in the infrastructure.

Virtuoso Fluid Guard Ring Developer Guide

Fluid Guard Ring Infrastructure

createVersion	<p>This slot is only a placeholder and has no effect on the FGR functionality.</p> <p><i>Data type:</i> string</p> <p>Its value is set as null string (" ").</p>
defComplementaryDevice	<p>The name of a complementary device.</p> <p>This slot is only a placeholder and has no effect on the FGR functionality.</p> <p><i>Data type:</i> string</p> <p>Its value is set as null string (" ").</p>
guardRingType	<p>The device is of N or P type.</p> <p>This slot is only a placeholder and has no effect on the FGR functionality. Its value is set as "N".</p>
designLib	<p>This slot is only a placeholder for now and has no effect on the FGR functionality.</p> <p><i>Data type:</i> string</p> <p>Its value is set as null string (" ").</p>
contAlignment	<p>This slot is only a placeholder for now and has no effect on the FGR functionality.</p> <p><i>Data type:</i> string</p> <p>Its value is set as null string (" ").</p>
cacheCreateVersion	<p>The create version of FGR sub-master.</p> <p><i>Data type:</i> string</p>
removeCornerContacts	<p>This slot is only a placeholder for now and has no effect on the FGR functionality.</p> <p><i>Data type:</i> string</p> <p>Its value is set as null string (" ").</p>
Slots used for drawing layers (<i>Data type:</i> string; <i>Default value:</i> nil)	
enclosingLayers	
metalLayer	
contLayer	
diffLayer	

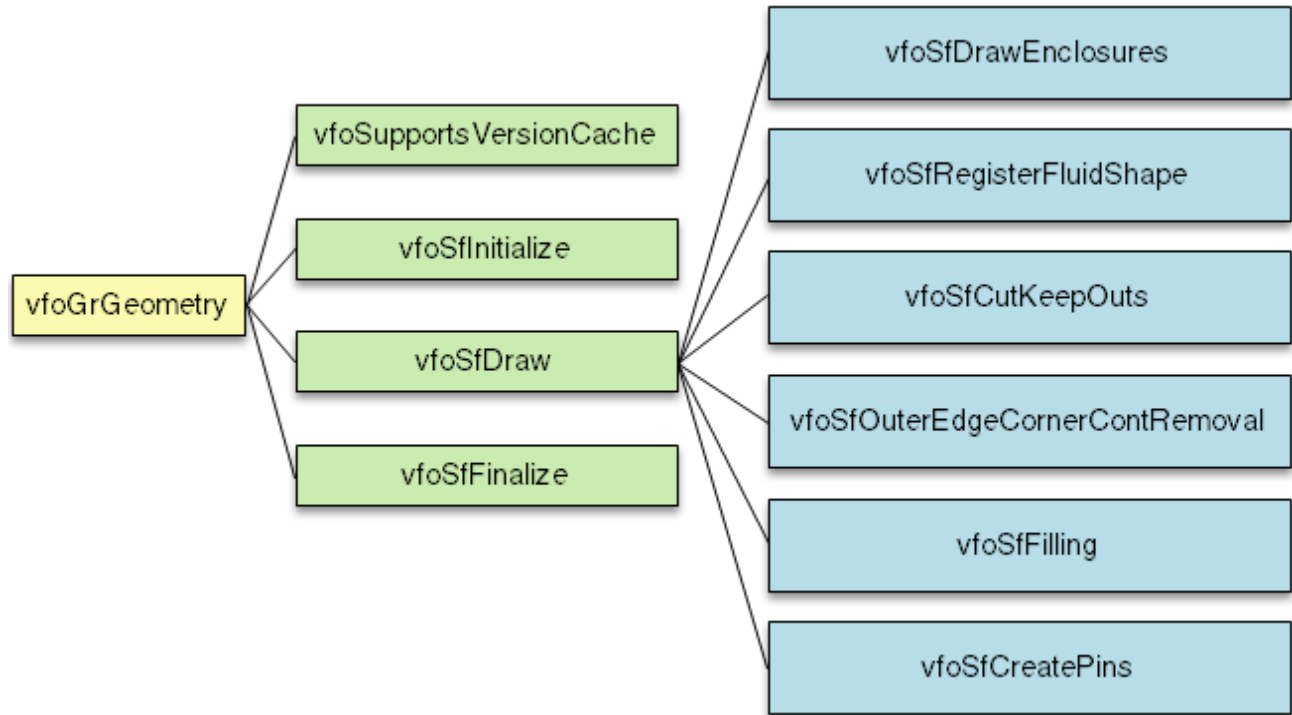
Virtuoso Fluid Guard Ring Developer Guide

Fluid Guard Ring Infrastructure

pinLayers
Slots for contact rules (<i>Data type: float; Default value: 0 . 0</i>)
xContWidth
yContWidth
xContSpacing
yContSpacing
xMetEnclCont
yMetEnclCont
xDiffEnclCont
yDiffEnclCont
metalEncCont1
metalEncCont2
diffEncCont1
diffEncCont2

Implementation Class Method Tree

Following is the method tree of the `vfoGuardRing` implementation class:



vfoAdvGuardRing

- [Slots](#)
- [Methods](#)

Slots

The `vfoAdvGuardRing` class has the following slots:

<code>techFile</code>	The name of the technology files. <i>Data type:</i> string
<code>vfoGRImpl</code>	The name of the implementation class. <i>Data type:</i> string
<code>classVersion</code>	The version of the VFO source code. <i>Data type:</i> integer

Virtuoso Fluid Guard Ring Developer Guide

Fluid Guard Ring Infrastructure

formalVersion	This slot is used only for internal processing in the infrastructure.
guardRingType	The device is of N or P type. This slot is only a placeholder and has no effect on the FGR functionality. Its value is set as "N".
Slots used for pitch support (<i>Data type: float</i>)	
verticalPitch	The fluid shape pitch in the vertical direction.
horizontalPitch	The fluid shape pitch in the horizontal direction.
horizontalSegWidth	The fluid shape width of the horizontal segments.
verticalSegWidth	The fluid shape width of the vertical segments.

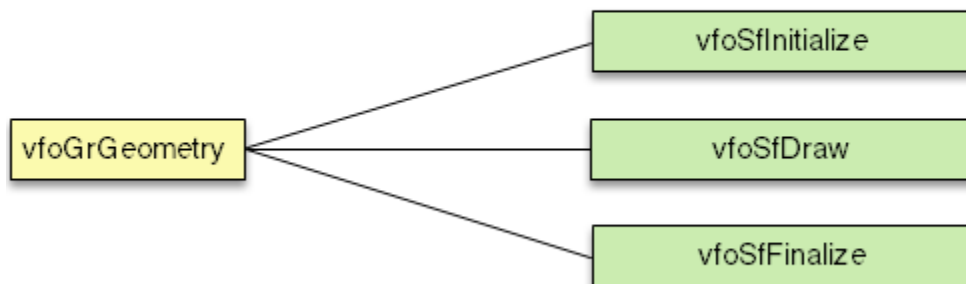
Methods

The `vfoAdvGuardRing` class has the following methods:

- `vfoGRCompareParams ((obj vfoAdvGuardRing) inst1 inst2)`
- `vfoAdvGRUpdateCDF ((obj vfoAdvGuardRing) libName dev view)`
- `vfoGRGetExtraArgumentName ((obj vfoAdvGuardRing))`
- `vfoSfInitialize ((pcell vfoAdvGuardRing))`
- `vfoSfDraw ((pcell vfoAdvGuardRing))`
- `vfoSfFinalize ((pcell vfoAdvGuardRing))`
- `vfoGRGeometry ((pcell vfoAdvGuardRing))`

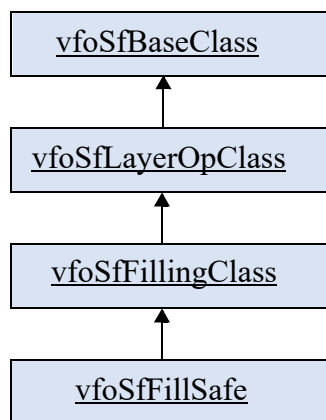
Implementation Class Method Tree

Following is the method tree of the `vfoAdvGuardRing` implementation class:



Filling Class

The filling class structure has the following classes:



Contact filling is controlled by the `vfoSfFillSafe` class.

`vfoSfLayerOpClass`

- Slots
- Method

Virtuoso Fluid Guard Ring Developer Guide

Fluid Guard Ring Infrastructure

Slots

The `vfoSfLayerOpClass` class has the following slots:

<code>layerop</code>	The description of the derived layer. For example: '(and ("M1" "drawing") ("M2" "drawing"))'
<code>pcell</code>	The object.
<code>lpp</code>	The layer-purpose pairs (LPPs).

Method

The `vfoSfLayerOpClass` class has the following method:

```
vfoSfDerive ((op vfoSfLayerOpClass) cv tmpLpp)
```

vfoSfFillingClass

- Slots
- Methods: *None*

Slots

The `vfoSfFillingClass` class has the following slots:

<code>cutLpp</code>	The LPP to be used as the contact or via layer.
<code>grid</code>	The grid to which the contact cut corners are restricted to.
<code>cw</code>	The contact width in x and y direction.
<code>cs_x</code>	The minimum x-spacing required between contact cuts.
<code>cs_half_x</code>	The minimum x-spacing required between contact cuts should be half of <code>cs_x</code> or half of <code>cs_x</code> ceiled to grid if it is off-grid.
<code>cs_y</code>	The minimum y-spacing required between contact cuts.
<code>cs_half_y</code>	The minimum y-spacing required between contact cuts should be half of <code>cs_y</code> or half of <code>cs_y</code> ceiled to grid if it is off-grid.
<code>overlap_x</code>	The minimum x-distance from edge of layer (LPP) to the first contact cut.

Virtuoso Fluid Guard Ring Developer Guide

Fluid Guard Ring Infrastructure

<code>overlap_y</code>	The minimum y-distance from edge of layer (LPP) to the first contact cut.
<code>fillStyle</code>	The value of the <code>?gap</code> argument of the <code>rodFillBBoxWithRects</code> SKILL function.

vfoSfFillSafe

- Slots: *None*
- Methods

Methods

The `vfoSfFillSafe` class has the following methods:

- `vfoSfFillShapes ((fillObj vfoSfFillBetter) shapes)`

This is the main method for filling contacts in fluid shapes.

- `vfoSfFillShapes_By_Path_Poly_Fill ((fill_obj vfoSfFillSafe) shapes)`

In this method, a fixed width path is filled using sub-rectangles and then extra rectangles are removed. In case of a polygon, rectangles are used for filling the box and then extra rectangles are removed. This method is fast, but rectangles falling on the sized polygon will be removed.

- `vfoSfFillShapes_By_Decomposition ((fill_obj vfoSfFillSafe) shapes)`

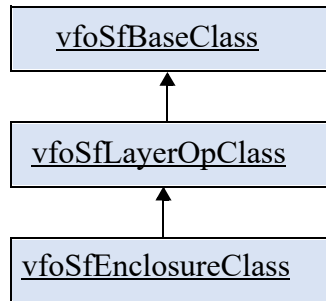
In this method, the shape is decomposed into rectangles and then those rectangles are filled with contacts or sub-rectangles.

- `vfoSfFillRects ((fill_obj vfoSfFillSafe) l_rects)`

In this method, `l_rects` is assumed to be a list of rectangles (`dbObjects`) that are already sorted by area such that the largest appears first.

Enclosure Class

The enclosure class structure has the following classes:



`vfoSfEnclosureClass` should be used to implement the drawing of enclosure layers.

vfoSfEnclosureClass

- Slots
- Method

Slots

The `vfoSfEnclosureClass` class has the following slots:

enclosure	The enclosure value.
coverInterior	The enclosure layer should have 'holes' or not.

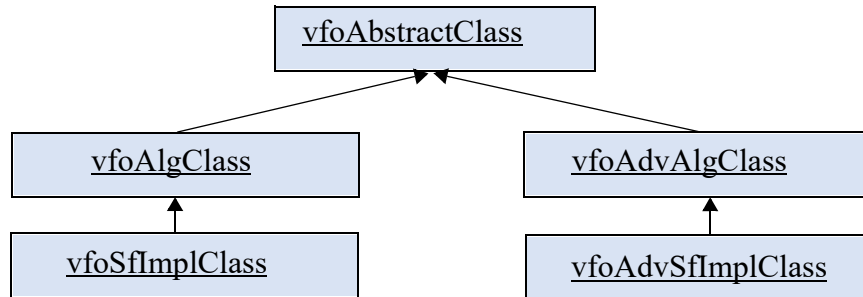
Method

The `vfoSfEnclosureClass` class has the following method:

- `vfoSfDerive ((enc vfoSfEnclosureClass) cv tmpLpp)`

Protocol Class

The protocol class structure has the following classes:



The edit operations on an FGR are controlled by protocol classes, `vfoAlgClass` and `vfoAdvAlgClass`. The `vfoAdvAlgClass` is used for pitch support.

vfoAbstractClass

- Slots: *None*
- Methods

Methods

The `vfoAbstractClass` class has the following methods to define the abstract interface for the functions that are required for enabling the editing operations in a fluid shape:

Method to Get Shape Data

<code>vfoGetShapeData(obj inst)</code>	Returns the <code>vfoShapeData</code> object, that is, path, rectangle, or polygon.
--	---

Methods to Enable or Disable an Edit Command

<code>vfoSupportsChop?(obj inst)</code>	Enables or disables the chop operation.
<code>vfoSupportsMerge?(obj inst)</code>	Enables or disables the merge operation.
<code>vfoSupportsCreateObstruction?(obj inst)</code>	
	Enables or disables the tunnel operation.
<code>vfoSupportsDeleteObstruction?(obj inst)</code>	

Virtuoso Fluid Guard Ring Developer Guide

Fluid Guard Ring Infrastructure

	Enables or disables the heal operation.
<code>vfoSupportsConvertToPolygon?(obj inst)</code>	
	Enables or disables the convert to polygon operation.
<code>vfoSupportsAbut?(obj inst)</code>	Enables or disables abutting.
<code>vfoSupportsUpdateModelShape?(obj inst)</code>	
	Enables or disables shapeData update.

Methods to Write the Function of an Edit Command

<code>vfoChopInstance(obj inst shapeData)</code>	
	Defines the chop function.
<code>vfoMergeInstances(obj inst1 inst2)</code>	
	Defines the merge function.
<code>vfoCreateObstruction(obj inst lpp points)</code>	
	Defines the tunnel function.
<code>vfoDeleteObstruction(obj inst lpp {point nil bBox ptList})</code>	
	Defines the heal function.
<code>vfoConvertToPolygon(obj inst)</code>	Defines the convert to polygon function.
<code>vfoAbut(obj instlist)</code>	Defines the abut function.
<code>vfoUpdateModelShape(obj inst shape newShapeType newPointList)</code>	
	Defines the shapeData update function.

vfoAlgClass

- Slots: *None*
- Method

Method

`vfoAlgClass` has the following method, which returns the `vfoShapeData` object, that is, path, rect, or polygon:

Virtuoso Fluid Guard Ring Developer Guide

Fluid Guard Ring Infrastructure

`vfoGetShapeData(obj inst)`

vfoSfImplClass

- Slots: *None*
- Methods: *None*

Note: `vfoSfImplClass` can only be extracted from `vfoAlgClass`.

vfoAdvAlgClass

- Slots: *None*
- Methods

Methods

The `vfoAdvAlgClass` class has the following methods:

Method to Get Shape Data

<code>vfoGetShapeData(obj inst)</code>	Returns the <code>vfoShapeData</code> object.
--	---

Methods to Enable or Disable an Edit Command

<code>vfoSupportsChop?(obj inst)</code>	Enables or disables the chop operation.
<code>vfoSupportsMerge?(obj inst)</code>	Enables or disables the merge operation.
<code>vfoSupportsCreateObstruction?(obj inst)</code>	
	Enables or disables the tunnel operation.
<code>vfoSupportsDeleteObstruction?(obj inst)</code>	
	Enables or disables the heal operation.
<code>vfoSupportsConvertToPolygon?(obj inst)</code>	
	Enables or disables the convert to polygon operation.
<code>vfoSupportsAbut?(obj inst)</code>	Enables or disables abutting.
<code>vfoSupportsUpdateModelShape?(obj inst)</code>	

Virtuoso Fluid Guard Ring Developer Guide

Fluid Guard Ring Infrastructure

	Enables or disables shapeData update.
--	---------------------------------------

Methods to Write the Function of an Edit Command

<code>vfoChopInstance(obj inst shapeData)</code>	
	Defines the chop function.
<code>vfoMergeInstances(obj inst1 inst2)</code>	
	Defines the merge function.
<code>vfoCreateObstruction(obj inst lpp points)</code>	
	Defines the tunnel function.
<code>vfoDeleteObstruction(obj inst lpp {point nil bBox ptList})</code>	
	Defines the heal function.
<code>vfoConvertToPolygon(obj inst)</code>	Defines the convert to polygon function.
<code>vfoAbut(obj instlist)</code>	Defines the abut function.
<code>vfoUpdateModelShape(obj inst shape newShapeType newPointList)</code>	
	Defines the shapeData update function.

vfoAdvSfImplClass

- Slots: *None*
- Methods: *None*

Note: `vfoAdvAlgClass` and `vfoAdvSfImplClass` are classes for pitch support.

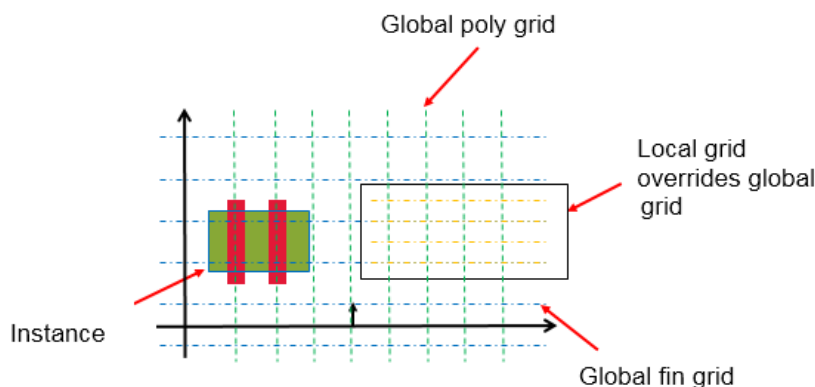
Pitch Support in Fluid Guard Ring

Overview of Pitch and Grid

Virtuoso has been enhanced in order to help layout engineers working on FinFET processes to improve their productivity. A new construct, `snapPatternDef`, has been defined in the technology file to capture width and spacing rules of layers in FinFET processes that have a grid-like nature. For more information, see [snapPatternDefs](#).

The `snapPatternDefs` are defined in the technology file. When designing custom FGR, the VFO infrastructure is not aware of the drawn layers. Hence, the snap pattern information in the technology file or layout canvas cannot be associated to the FGR. So, for the fluid shape to adhere to the `snapPatternDefs`, four new parameters have been introduced. They are `horizontalPitch`, `verticalPitch`, `horizontalSegWidth`, and `verticalSegWidth`.

The example below illustrates a simple `snapPatternDef` specifying global and local, fin and poly grids.



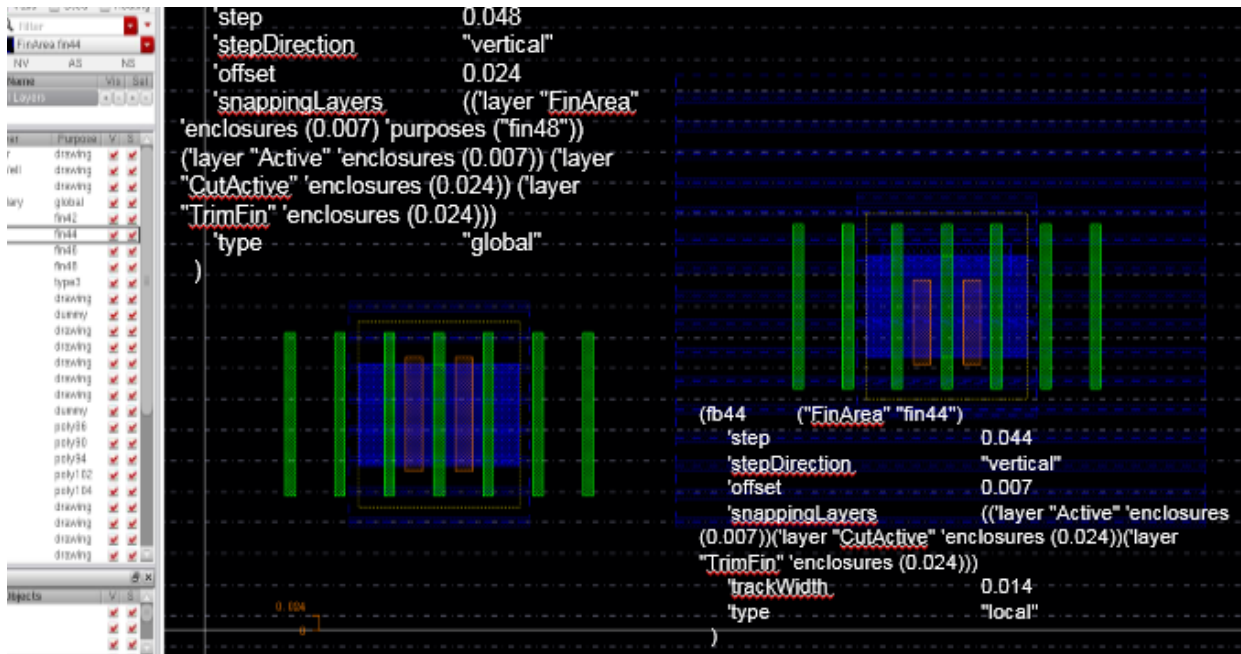
In the above example, the global fin grid starts at the origin axis, or PR boundary, if the PR boundary does not coincide with the origin axis, plus the offset value. Also, if there is a snap

Virtuoso Fluid Guard Ring Developer Guide

Pitch Support in Fluid Guard Ring

pattern, the instance is snapped using the snap pattern, else it is snapped using the shape in the snapping layer.

The example below illustrates a `snapPatternDef` specifying global and local fin grids.



Calculating the Fluid Shape Data using Pitch Parameters

Pitch support has been added for horizontal and vertical direction. For this functionality, four CDF and formal parameters, `horizontalPitch`, `verticalPitch`, `horizontalSegWidth`, and `verticalSegWidth`, have been added. The parameters, `horizontalPitch` and `verticalPitch`, map layer-specific pitch values to the fluid shape pitch in horizontal and vertical directions. Typically, `horizontalPitch` is mapped to poly pitch and `verticalPitch` is mapped to fin pitch. Apart from the pitch parameters, `horizontalSegWidth` and `verticalSegWidth` parameters are also required to calculate accurate fluid shape segment lengths. This is because the segment widths are mapped to a number of poly lines and fins. Also, the segment lengths for fluid shape depends on number of poly lines and fins, as described in the next section.

Currently, pitch support has been added for the path shape type. A path has a centerline and width. Pitch calculations for center line and width are described below.

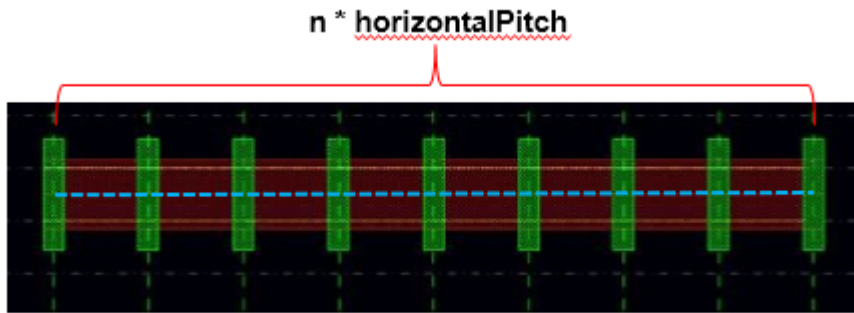
Calculating the Centerline of Path Style Fluid Shape using Pitch Parameters

This section describes how centerline for path style fluid shape is calculated using pitch parameters. The path style fluid shape with no corner, one corner, and two corners are described below.

■ No corner

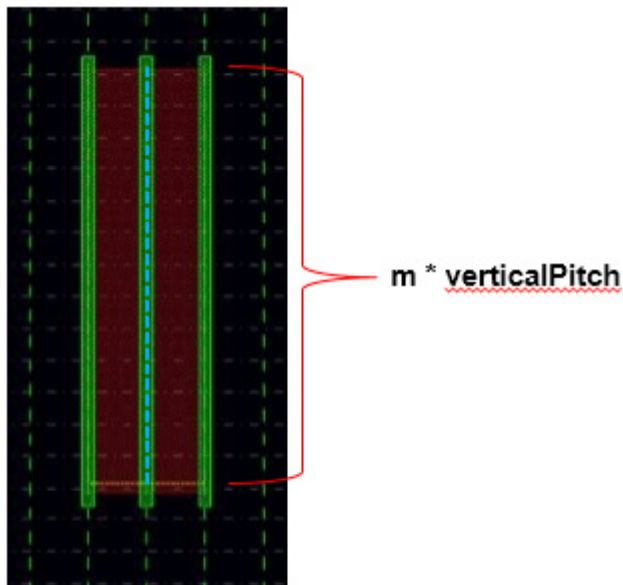
Horizontal Segment Length = $n * \text{horizontalPitch}$

n is the pitch multiplier



Vertical Segment Length = $m * \text{verticalPitch}$

m is the pitch multiplier



Virtuoso Fluid Guard Ring Developer Guide

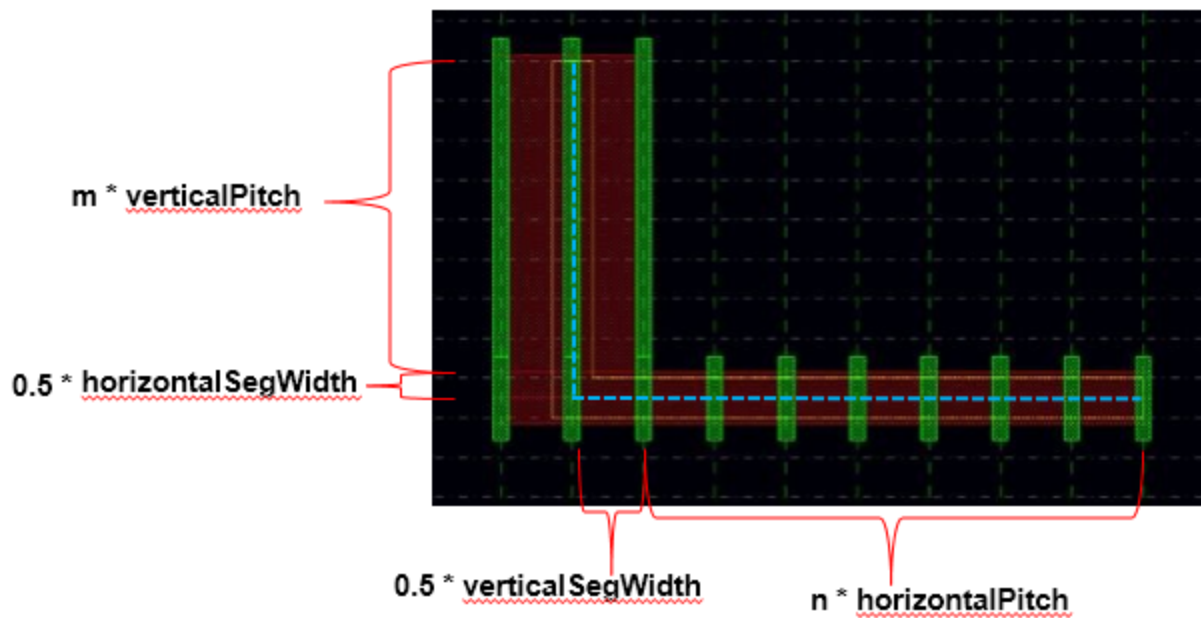
Pitch Support in Fluid Guard Ring

■ One corner

Horizontal Segment Length = $n * \text{horizontalPitch} + 0.5 * \text{verticalSegWidth}$

Vertical Segment Length = $m * \text{verticalPitch} + 0.5 * \text{horizontalSegWidth}$

n and m are the pitch multipliers



Virtuoso Fluid Guard Ring Developer Guide

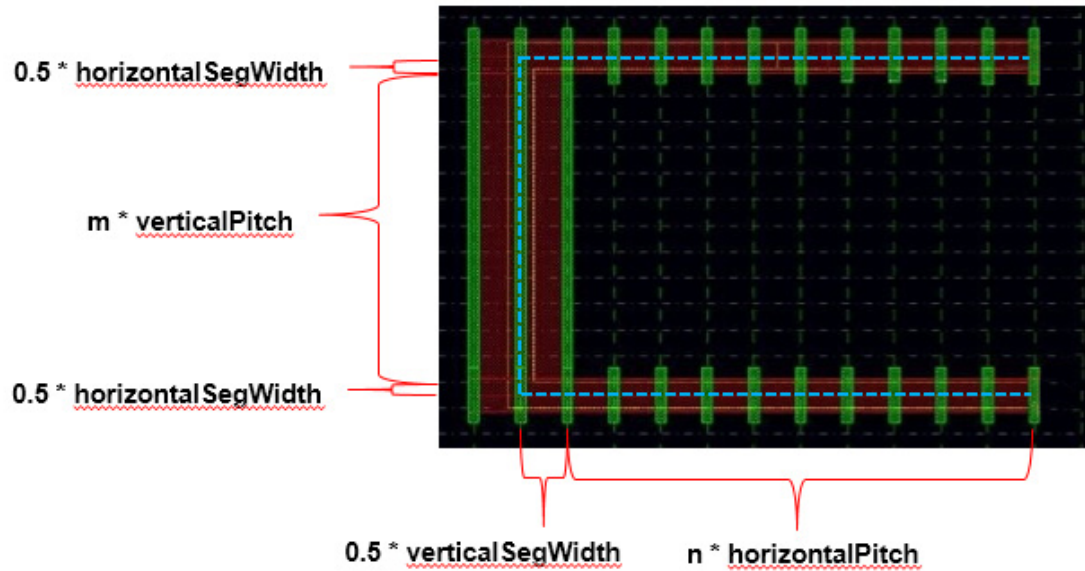
Pitch Support in Fluid Guard Ring

■ Two Corners (Open Ends)

Horizontal Segment Length = $n * \text{horizontalPitch} + \text{verticalSegWidth}$

Vertical Segment Length = $m * \text{verticalPitch} + \text{horizontalSegWidth}$

n and m are the pitch multiplier



Virtuoso Fluid Guard Ring Developer Guide

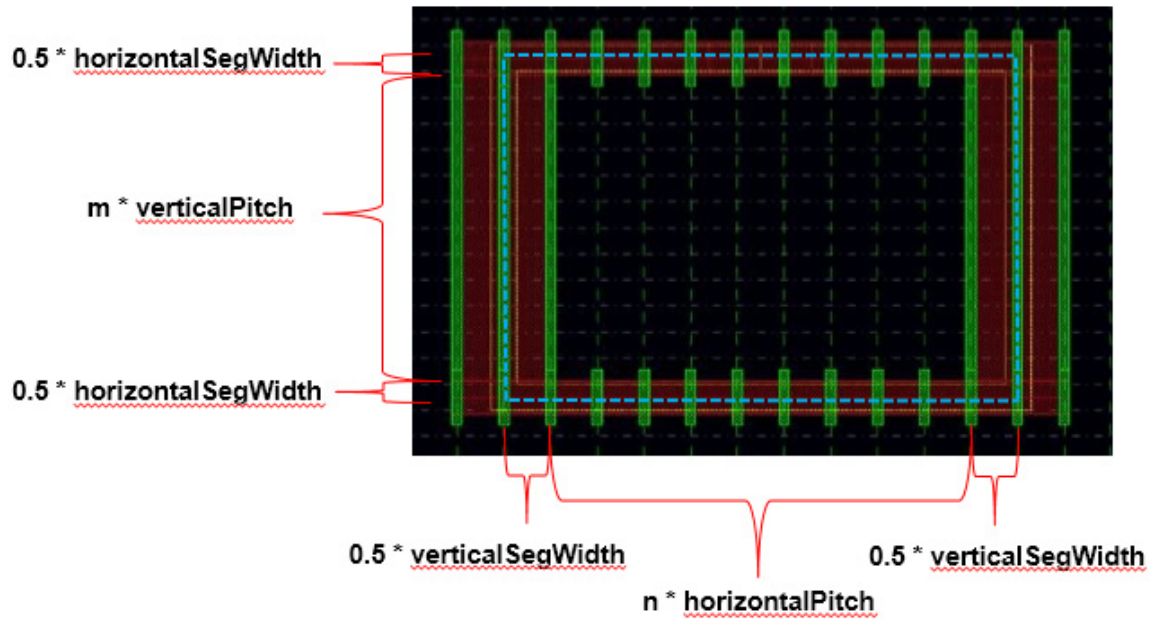
Pitch Support in Fluid Guard Ring

■ Two Corners (Ring Style)

Horizontal Segment Length = $n * \text{horizontalPitch} + \text{verticalSegWidth}$

Vertical Segment Length = $m * \text{verticalPitch} + \text{horizontalSegWidth}$

n and m are the pitch multiplier

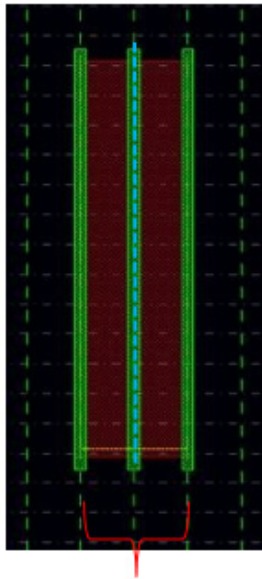


Calculating the Width of Path Style Fluid Shape using Pitch Parameters

The path width for path style fluid shape is calculated as described below:

■ Horizontal

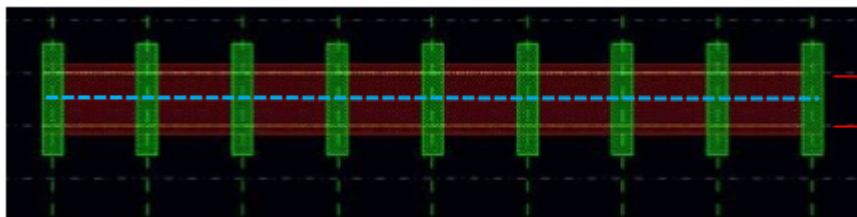
`pathWidth = horizontalSegWidth`



horizontalSegWidth

■ Vertical

`pathWidth = verticalSegWidth`



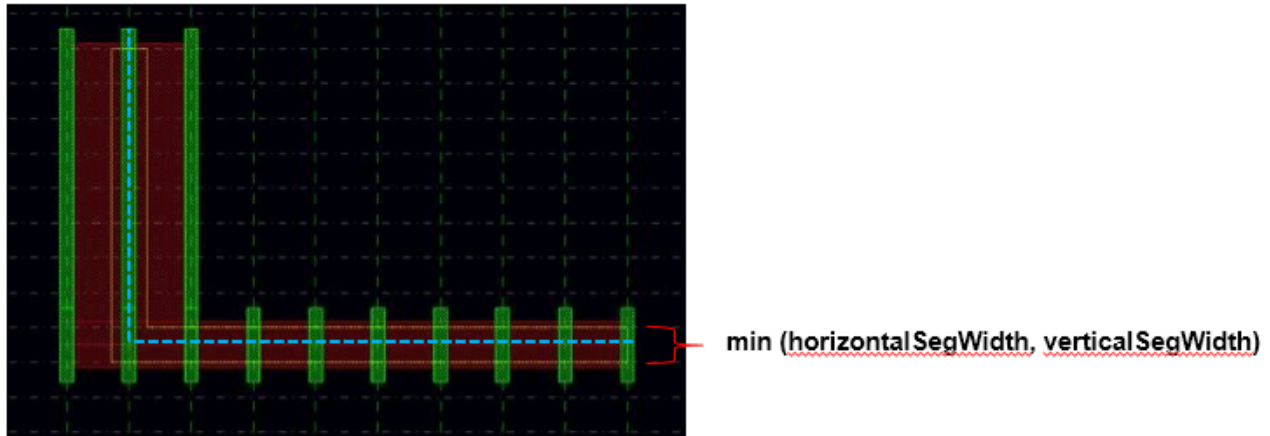
verticalSegWidth

Virtuoso Fluid Guard Ring Developer Guide

Pitch Support in Fluid Guard Ring

■ Rectangular or Rectilinear

`pathWidth = min (horizontalSegWidth, verticalSegWidth)`



Note: When creating a fluid shape, if the data points you specify are such that the dimension is less than `(horizontalPitch + verticalSegWidth)` or `(verticalPitch + horizontalSegWidth)`, the VFO infrastructure snaps the shape point to the closest valid point, as per the calculations provided in the [Calculating the Centerline of Path Style Fluid Shape using Pitch Parameters](#) section.

Develop and Define a Fluid Guard Ring Device

Customizing an FGR involves more than one step depending on its design: extending the classes and their methods provided in VFO infrastructure, defining the device and its properties in technology file, adding or updating CDF parameters.

This chapter covers these aspects in detail in the following sections:

- [Extending the VFO Infrastructure](#)
- [Defining Fluid Guard Ring Devices](#)
- [Adding or Modifying CDF Parameters](#)

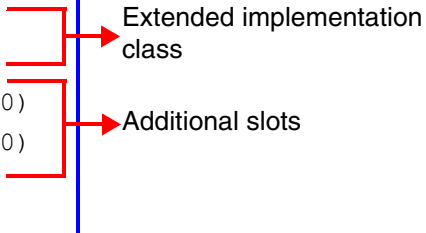
Extending the VFO Infrastructure

The implementation, filling, and protocol classes in the VFO infrastructure are extendible as explained in the sections below.

Extending the Implementation Class

While writing code for a customized FGR device, a PDK developer extends a class from the `vfoAdvGuardRing` implementation class. This extended class inherits the slots from base classes and new slots can be defined in it. The examples below illustrate how to do this.

```
(defclass sub1GuardRing (vfoAdvGuardRing)
  (
    (userParam1 @initarg userParam1 @initform 0.0)
    (userParam2 @initarg userParam2 @initform 0.0)
    ...
  ))
```



Extended implementation class

Additional slots

Virtuoso Fluid Guard Ring Developer Guide

Develop and Define a Fluid Guard Ring Device

To specify the drawing method for a new FGR device, override the `vfoGRGeometry()` method and/or its relevant sub-methods by passing an object of the custom class, as shown below. The example below also shows how to override `vfoGRGeometry()` and `vfoSfDraw()` methods.

```
vfoGRGeometry(obj sublGuardRing)
{
    ...
    vfoSfInitialize(obj)
    vfoSfDraw(obj)
    vfoSfFinalize(obj)
    ...
}
```

Override `vfoGRGeometry` by passing an object of extended implementation class, `sublGuardRing`

```
(defmethod vfoSfDraw ((obj sublGuardRing))
  dbCreateRect(obj->cv "layer1" '((3 3) (8 8)))
  ...)
```

Write your own code for `sublGuardRing`

Note: In case you are implementing fluid guard ring without pitch support, extend from `vfoGuardRing` class.

Extending the Filling Class

To customize the filling of the contacts in an FGR, extend the `vfoSfFillSafe` filling class and override the relevant methods.

In the example below, `subFillClass` is the extended filling class and the `vfoSfFillShapes()` method is overridden to contain the custom contact filling code.

```
(defclass subFillClass (vfoSfFillSafe)
  (...)
)
```

Extended filling class, `subFillClass`

```
(defmethod vfoSfFillShapes((fillObj subFillClass)
  shapes)
  dbCreateRect(fillObj->cv "layer1" '((3 3) (8 8)))
  callNextMethod()
  ...
)
```

Update the `vfoSfFillShapes` method and pass an object of `subFillClass`

Defining Fluid Guard Ring Devices

The process of defining FGR devices requires definition of the device class, declaration of the device, and specification of the device properties in the technology file. The updated FGR device definitions are reflected after you save and reload the edited technology file.

Defining the Device Class


While developing a customized FGR device, the `tcCreateDeviceClass` class in the technology file acts as a template for device definition. Add the new formal parameters to this device class, as shown below.

Existing

```
tcCreateDeviceClass("layout"
"cdsGuardRing"
    ; class parameters
...)
```

Updated

```
tcCreateDeviceClass( "layout"
"cdsGuardRing"
    ; class parameters
((userParam1 0.0) (userParam2 0.0)
(userParam3 0.0)
...)
```



New formal parameters

You can define multiple such device definitions. In this case, the class parameter is a union of all parameters belonging to different devices of the same device class.

Note: For each formal parameter, you must define a CDF parameter with the same name. For example, when you define a formal parameter, `myNewParam`, also create a CDF parameter named `myNewParam`. Refer to the [Defining Fluid Guard Ring Devices](#) section for details.

Example Device Class Definition

Following is an example of a complete device class definition:

```
tcCreateDeviceClass( "layout" "cdsGuardRing"
    ; class parameters
(    (pinName "B1")
      (defComplementaryDevice nil)
      (classVersion nil)
      (vfoGRImpl "vfoAdvGuardRing")
      (enclosureClass "vfoSfEnclosureClass")
```

Virtuoso Fluid Guard Ring Developer Guide

Develop and Define a Fluid Guard Ring Device

```
(vfoProtocolClass "vfoAdvSfImplClass")
  (highlightLpp (quote ("annotate" "drawing")))
  (mainLpp (quote ("y2" "drawing")))
  (modelLpp (quote ("y0" "drawing")))
  (metalLayer nil)
  (contLayer nil)
  (tmpLpp (quote ("instance" "drawing")))
  (diffLayer nil)
  (guardRingType nil)
  (termName "B")
)
; formal parameters
(
  (contAlignment "")
  (xDiffEnclCont 0.0)
  (yDiffEnclCont 0.0)
  (xMetEnclCont 0.0)
  (yMetEnclCont 0.0)
  (xContSpacing 0.0)
  (yContSpacing 0.0)
  (xContWidth 0.0)
  (yContWidth 0.0)
  (enclosingLayers "nil")
  (shapeData "nil")
  (shapeType "none")
  (decompositionMode (if (getShellEnvVar "FGR_USE_ALIGNCUTS") "alignCuts"
"fill45-path-poly"))
  (hide_keepouts t)
  (fillStyle "distribute")
  (fillClass "vfoSfFillSafe")
  (debug 0)
  (do_something t)
  (formalVersion 0)
  (keepOuts nil)
  (removeCornerContacts nil)
  (horizontalSegWidth 0.0)
  (verticalSegWidth 0.0)
  (verticalPitch 0.0)
  (horizontalPitch 0.0)
)
; IL codes specifying geometry
```

Virtuoso Fluid Guard Ring Developer Guide

Develop and Define a Fluid Guard Ring Device

```
(eval
  (quote
    (if
      (vfoIsSuperMaster tcCellView)
      (progn
        (dbCreateLabel tcCellView modelLpp
          (range 0 0) "superMaster"
          "lowerLeft" "R0" "roman" 1.0
        )
        (vfoSetProtocolClassName tcCellView
          (concat vfoProtocolClass)
        )
      )
    )
    (let
      ((result
        (errset
          (when
            t
            (vfoGRGeometry
              (makeInstance
                (or
                  (findClass
                    (concat vfoGRImpl "_ver_" formalVersion)
                  )
                  (error "SKILL Class %L does not exist:"
                    (concat vfoGRImpl "_ver_" formalVersion)
                  )
                )
              )
            )
            ?cv tcCellView ?keepOuts keepOuts ?formalVersion formalVersion
          )
        )
        ?do_something do_something
        ?debug debug ?fillClass fillClass ?fillStyle fillStyle
        ?hide_keepouts hide_keepouts ?decompositionMode decompositionMode
        ?shapeType shapeType ?shapeData shapeData ?enclosingLayers
        enclosingLayers
        ?xContWidth xContWidth ?yContWidth yContWidth ?xContSpacing
        xContSpacing ?yContSpacing yContSpacing
        ?xMetEnclCont xMetEnclCont ?yMetEnclCont yMetEnclCont
        ?xDiffEnclCont xDiffEnclCont
        ?yDiffEnclCont yDiffEnclCont ?contAlignment contAlignment
        ?removeCornerContacts removeCornerContacts
        ?vfoGRImpl vfoGRImpl ?modelLpp modelLpp ?tmpLpp tmpLpp
        ?highlightLpp highlightLpp
```


Virtuoso Fluid Guard Ring Developer Guide

Develop and Define a Fluid Guard Ring Device

```
(enclosureClass "vfoSfEnclosureClass")
(vfoProtocolClass "vfoAdvSfImplClass")
(hilighLpp (quote ("annotate" "drawing")))
(vfoGRImpl "sublGuardRing")
(mainLpp (quote ("Active" "drawing")))
(modelLpp (quote ("y0" "drawing")))
(tmpLpp (quote ("instance" "drawing")))
(guardRingType "N")
(termName "FGRTerm")
(pinName "FGRPin")
(defComplementaryDevice "")
)
( (shapeData "nil")
  (shapeType "none")
  (decompositionMode "fill45-path-poly")
  (hide_keepouts t)
  (fillStyle "distribute")
  (fillClass "vfoSfFillSafe")
  (debug 0)
  (do_something t)
  (formalVersion 0)
  (keepOuts (quote nil))
  (horizontalSegWidth 0.048)
  (verticalSegWidth 0.172)
  (verticalPitch 0.048)
  (horizontalPitch 0.086)
)
)
```

Here, replace the names of the implementation, protocol, and filling classes with the names of corresponding new classes that you have extended for the custom FGR. Then, add the new formal parameters in this device declaration construct.

Virtuoso Fluid Guard Ring Developer Guide

Develop and Define a Fluid Guard Ring Device


Therefore, if we continue with the examples of class extendibility given in the sections above, the `tcDeclareDevice` construct will need to be updated as shown below.

Existing

```
tcDeclareDevice( "layout"
"cdsGuardRing" "myFluidGuardRing"
  ( (classVersion 1)
    (vfoProtocolClass
      "vfoSfImplClass") ...
    vfoGRImpl "vfoGuardRing") ...
    (modelLpp (quote "y0"
      "drawing"))) ...)
  ( (shapeData "nil")
    (shapeType "none") ...
    (fillClass "vfoSfFillSafe") ...
  ...)
```

Updated

```
tcDeclareDevice( "layout"
"cdsGuardRing" "myFluidGuardRing"
  ( (classVersion 1)
    (vfoProtocolClass
      "sub1EditClass") ...
    vfoGRImpl "sub1GuardRing") ...
    (modelLpp (quote "y2"
      "drawing"))) ...)
  ( (shapeData "nil") (shapeType
    "none") ...
    (fillClass "subFillClass") ...
    ((userParam1 0.01)
      (userParam2 0.01)
      (userParam3 0.05)
    ...)
  ...)
```

 New formal parameters

Specifying Device Properties

The `tfcDefineDeviceProp` construct of the technology file defines the device properties. Following is an example of this construct:

```
tfcDefineDeviceProp(
; (viewName      deviceName      propName      propValue)
  (layout        "myFluidGuardRing"  vfoProtocolClass  "vfoSfImplClass")
)
```

Within this construct, the following properties are available for handling customized FGR devices:

Property name	Description
---------------	-------------

Virtuoso Fluid Guard Ring Developer Guide

Develop and Define a Fluid Guard Ring Device

<code>vfoProtocolClass</code>	Defines implementation protocol class that contains the editing commands available for the corresponding FGR device. Use this property to replace the name of the existing implementation protocol class with the one created to override the edit operations. Acceptable values: Any <i>string</i> type value								
<code>vfoGRHideDeviceInForms</code>	Controls the display of devices on the following two forms – Install Guard Ring and Create Guard Ring. Acceptable values: One of the following <i>string</i> type values: <table><tr><th>Value</th><th>Hides the specified device from</th></tr><tr><td><code>installAndCreateForm</code></td><td>Install Guard Ring form Create Guard Ring form</td></tr><tr><td><code>installForm</code></td><td>Install Guard Ring form</td></tr><tr><td><code>createForm</code></td><td>Create Guard Ring form</td></tr></table>	Value	Hides the specified device from	<code>installAndCreateForm</code>	Install Guard Ring form Create Guard Ring form	<code>installForm</code>	Install Guard Ring form	<code>createForm</code>	Create Guard Ring form
Value	Hides the specified device from								
<code>installAndCreateForm</code>	Install Guard Ring form Create Guard Ring form								
<code>installForm</code>	Install Guard Ring form								
<code>createForm</code>	Create Guard Ring form								

For example, the following `tfcDefineDeviceProp` construct shows how to define custom protocol class, `sub1EditClass`, and hide the device named `myFluidGuardRing` from the Install Guard Ring form:

```
tfcDefineDeviceProp(  
; (viewName      deviceName      propName      propValue)  
  (layout        "myFluidGuardRing" vfoProtocolClass "sub1EditClass")  
  (layout        "myFluidGuardRing" vfoGRHideDeviceInForms "installForm")  
)
```

Adding or Modifying CDF Parameters

To manage the geometries of the customized FGR device, you can modify the existing component description format (CDF) parameters and attributes, or add new ones by following the steps mentioned below.

1. Create or define new formal parameters for the customized FGR device as given below.
 - a. Dump the ASCII technology file.
 - b. Search for the definition of the FGR device class in the technology file.

Virtuoso Fluid Guard Ring Developer Guide

Develop and Define a Fluid Guard Ring Device

- c. Add the new formal parameters to the FGR device class, `tcCreateDeviceClass`, and add a list of the same parameters in the `vfoGRGeometry` method.
- d. Search for the FGR device declaration in the technology file that needs to be customized and replace the name of the implementation class, `vfoAdvGuardRing`, with the name of your new implementation class.
- e. Add the new formal parameters in the updated FGR device declaration construct.
- f. Save and load the edited ASCII technology file to update the FGR device definition in the technology database.
- g. Extend the new FGR implementation class from the `vfoAdvGuardRing` class and define the new formal parameters in the following format:

```
defclass (subImplClass (vfoAdvGuardRing)
  (
    (newParam1 @initarg newParam1 @initform value)
    (newParam2 @initarg newParam2 @initform value)
  )
)
```

Note: This step is not required when you are updating an existing CDF parameter.

2. Add or update CDF parameters for the customized FGR device as given below.

- a. Write a procedure that defines the `vfoUpdateCDF(lib cell view)` trigger that helps to add new CDF parameters and update the existing ones. When the implementation class of a device is other than `vfoAdvGuardRing`, this trigger gets called while loading the technology file.
- b. Make sure you assign read and write access rights to the new CDF parameters using the `reader` and `writer` properties with the `cdfId` SKILL variable. For detailed information about `cdfId`, refer to the *Modifying Simulation Information* chapter in *Component Description Format User Guide*.

- c. Use the `cdfCreateParam` SKILL function to define the attributes of the new CDF parameters. For detailed information about `cdfCreateParam`, refer to the *CDF SKILL Summary* chapter in *Component Description Format User Guide*.

Note: While updating an existing CDF parameter, you can modify the values of the current attributes and add new attributes.

- d. Define a callback for your CDF parameter, if needed. Each user-defined CDF parameter should have a corresponding callback definition.
- e. Save the CDF definition using the `cdfSaveCDF` function. If the CDF description already exists, the old one gets overwritten by the new description.

Virtuoso Fluid Guard Ring Developer Guide

Develop and Define a Fluid Guard Ring Device

For detailed information about the triggers, SKILL functions, steps listed above, and related example, refer to the [Adding and Managing CDF Parameters for Fluid Guard Rings](#) application note available on the [Cadence Online Support](#) website.

Following example code provides a snapshot of adding new CDF parameter `userDefinedParam`, updating the existing CDF parameter `shapeData`, and saving them:

```
procedure(vfoUpdateCDF(libName cellName viewName)
  prog((cellId cdfId)
    cellId = ddGetObj(libName cellName)
    unless(cellId return())

    cdfId = cdfGetBaseCellCDF(cellId)
    unless(cdfId
      cdfId = cdfCreateBaseCellCDF(cellId)
    );;unless
;;Adding new CDF
    cdfCreateParam(cdfId
      ?name          "userDefinedParam"
      ?prompt        "user CDF"
      ?type          "boolean"
      ?defValue      t
      ?callback      "userDefinedParam_CdfCB()"
    )
;;Associating reader and writer properties with the new parameters
    fromcdf='(lambda (x) x)
    tocdf='(lambda (x) x)

    ;;Adding readers
    putprop(cdfId->readers fromcdf (concat "userDefinedParam"))

    ;;Adding writers
    putprop(cdfId->writers tocdf (concat " userDefinedParam"))
;;Updating existing CDF attribute
    foreach(param cdfId~>parameters
      when(param~>name == "shapeData"
        param~>display="nil"
        param~>callback="myShapeData_CdfCB()"
      )
    )
  )
;;Defining the myShapeData_CdfCB() callback:
procedure(myShapeData_CdfCB()
```

Virtuoso Fluid Guard Ring Developer Guide

Develop and Define a Fluid Guard Ring Device

```
    let(  
    printf("myShapeData Callback \n")  
    )  
)  
cdfSaveCDF(cdfId)
```

Customize Create Guard Ring Form

The Create Guard Ring form is an option-type form that facilitates interactive FGR creation. Triggers and SKILL APIs are available to enable you to customize this form to suit your custom FGR requirements. Once you select a *Device* from the list available on this form, values of the other fields are populated from the supermaster defaults.

The following types of customizations are possible on a Create Guard Ring form:

- Modifying the Existing Create Guard Ring Form
 - Adding New User-Defined GUI Components to the Form
 - Updating the Existing GUI Components on the Form
 - Updating Properties of User-Defined GUI Components
- Pitch Parameter Support in the Create Guard Ring Form
- Creating a New Create Guard Ring Form
- Using the Create Guard Ring Form

Modifying the Existing Create Guard Ring Form

The ability to modify the Create Guard Ring form provides you the control over the existing system-defined GUI components. It also enables you to add new user-defined GUI components, such as fields and buttons, and define callbacks for the specific ones you want to use on the form.

The sections below describe the Create Guard Ring form modification methodology.

Adding New User-Defined GUI Components to the Form

To add new user-defined GUI components to the Create Guard Ring form, you need to do the following:

Virtuoso Fluid Guard Ring Developer Guide

Customize Create Guard Ring Form

1. Write a procedure to define the `vfoGRAddCreateFormFields` trigger.
2. Use the `hiCreate*` functions to add new GUI components on the Create Guard Ring form.
3. Set the GUI component to a specific location on the form.

In the Create Guard Ring form, there are multiple global lists that enable you to add various form components. The common global list (also called common queue) defines the set of form components that are displayed on all four tabs (*Wrap*, *Path*, *Rect*, and *Polygon*) of this form. For example, the GUI components, such as *Technology*, *Device*, and *Contact Rows*, that are common on all tabs exist in the common queue area. To add a new component in this area, use the `vfoGRAddFieldsInCommonQ` queue.

However, if you want to update the GUI components on only a specific tab of the Create Guard Ring form, use the following queues:

- ☐ `vfoGRAddFieldsInPathTabQ` (use for the *Path* tab)
- ☐ `vfoGRAddFieldsInRectTabQ` (use for the *Rect* tab)
- ☐ `vfoGRAddFieldsInPolygonTabQ` (use for the *Polygon* tab)
- ☐ `vfoGRAddFieldsInWrapTabQ` (use for the *Wrap* tab)

4. Use the `vfoGRSetExtraArgument` (*associativeList*) SKILL function to make the data available for processing by `extraArguments` in the FGR infrastructure that resides in Virtuoso.

Each element of the associative list is a 'key value' pair, where the *key* is the name of the FGR device parameter and *value* is the value associated to it, that is,
((<FGR_device_parameter_name> <value_of_GUI_component>) ...)

Updating the Existing GUI Components on the Form

To update the GUI components that are currently displayed on the Create Guard Ring form, you need to do the following:

1. Write a procedure to define the `vfoGRUpdateCreateForm` (<formPointer>) trigger, where *formPointer* is a pointer to the Create Guard Ring form.
2. Hide existing default fields from the form by using the `vfoGRSetCreateFormAllFieldsInVisible` (flag) SKILL function.

Note: This SKILL function cannot be used to hide user-defined fields.

You can use the `vfoGetImplementationClassName` (<libName> <deviceName>) SKILL function to identify the implementation class of different

devices and hide the form fields only when the implementation class is not `vfoGuardRing`.

3. Reset the properties of the fields or components displayed on the form by using the `vfoGRSetCreateFormFieldProp(<promptName> <propertyName> <propertyValue>)` SKILL function. It supports the use of the following property values: `value`, `defValue`, `editable`, and `invisible`.

Updating Properties of User-Defined GUI Components

If you add a user-defined GUI component in the common queue, you can access its pointer using the `vfoGRGetCommonQPtr` SKILL function to get or set the properties of its component. For example, to set the property of a newly added form field, `UserSelectVertWidth`, you can write the following procedure:

```
procedure( setCreateFormUserFieldProp(actField promptString property value)
let(( newField )
    importSkillVar(vfoGRAddFieldsInCommonQ)
    evalstring( sprintf( nil "vfoGRGetCommonQPtr()->%s->%s = %s"
                          UserSelectVertWidth->hiFieldSym property value )
    )
))
```

Virtuoso Fluid Guard Ring Developer Guide

Customize Create Guard Ring Form

Example: Hiding Existing Fields and Adding New Check Box to the Form

Consider a scenario where you want to hide all pre-defined GUI components except *Technology* and *Device* fields on the Create Guard Ring form, and add a new check box, *User-defined Parameter* instead (as depicted in the image below).



For the required customization, perform the following steps:

1. Hide the existing form fields if the device selected is a custom FGR device.
 - a. Use `vfoGRGetCreateFormFieldProp` to read and get the component prompt value and property of the *Device* and *Technology* fields.
 - b. Use the `vfoGetImplementationClassName` SKILL function to identify the device implementation class.

2. Create a new user-defined Boolean field with prompt *User-defined Parameter* using the *hiCreateBooleanButton* function, add its name to the `vfoGRAddFieldsInCommonQ` list and specify its location.
3. Run the `vfoGRSetExtraArgument` SKILL function to make the values of the GUI components on the Create Guard Ring form available for processing by `extraArguments` in the FGR infrastructure.

For detailed information, refer to the [Customizing Create Guard Ring Form](#) application note available on the [Cadence Online Support](#) website.

Pitch Parameter Support in the Create Guard Ring Form

Corresponding to the pitch parameters, `horizontalPitch`, `verticalPitch`, `horizontalSegWidth`, and `verticalSegWidth`, new fields have been added to the Create Guard Ring Form. The field prompts are *Horizontal Pitch*, *Vertical Pitch*, *Horizontal Segment Width*, and *Vertical Segment Width*. By default, these fields are invisible and non-editable.

Horizontal Pitch provides the value of the horizontal direction pitch parameter and corresponds to the `horizontalPitch` formal parameter added in the FGR device definition.

Vertical Pitch provides the value of the vertical direction pitch parameter and corresponds to the `verticalPitch` formal parameter added in the FGR device definition.

Horizontal Segment Width is added to calculate the correct fluid shape segment length in the vertical direction. The value of this field should be a multiple of the value in the `horizontalPitch` parameter.

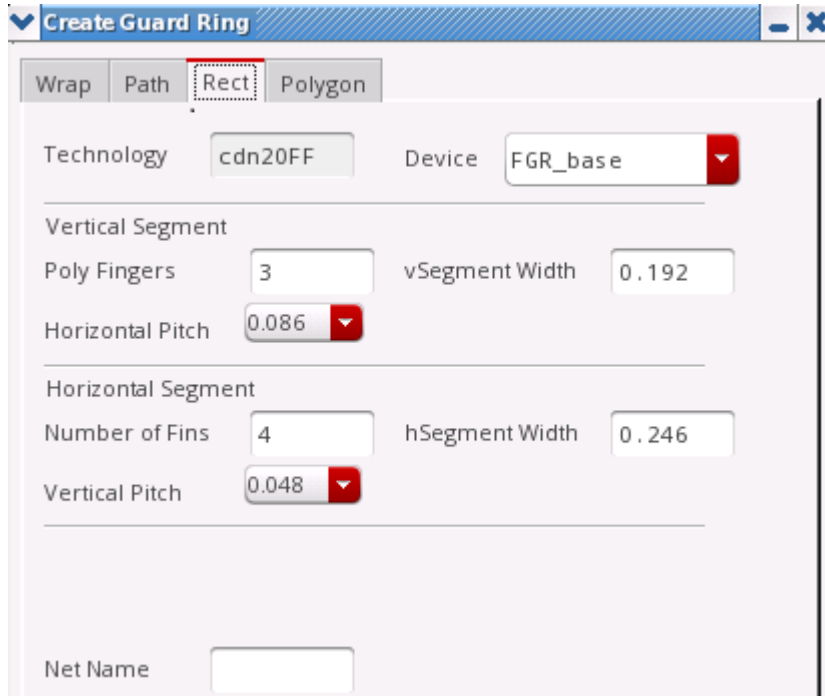
Vertical Segment Width is added to calculate the correct fluid shape segment length in the horizontal direction. The value of this field should be a multiple of the `verticalPitch` parameter.

The above fields are float type and accept only positive values. As these fields correspond to the VFO infrastructure parameters, these should not be exposed to the end-user. PDK developers should create new GUI fields to take inputs from the user and process the inputs

Virtuoso Fluid Guard Ring Developer Guide

Customize Create Guard Ring Form

using callbacks to provide value to infrastructure parameters. One example of the Create Guard Ring form is shown below:



Creating a New Create Guard Ring Form

The Create Guard Ring form launched from the layout editor is an options-type form that displays the *Hide*, *Cancel*, and *Defaults* buttons. However, if you prefer to use a standard-type form with *OK*, *Cancel*, and *Apply* buttons, you have the flexibility to create such a Create Guard Ring form as well. The standard-type form can be run from other Virtuoso applications, like Module Generator (Modgen) and Constraint Manager.

The following steps describe the Create Guard Ring form creation process:

1. Define a new form pointer along with unique fields and form symbol using the `vfoGRNewCreateForm` SKILL function.
2. Register the customization procedure using the `vfoGRRegCreateFormUpdateCallback` SKILL function.

The following is an example of the above steps:

```
; create the new Create Guard Ring form and store its form pointer
form_modgen = vfoGRNewCreateForm ("MODGEN" 'OKCancelApply)
; register the callback for the given form pointer
```

Virtuoso Fluid Guard Ring Developer Guide

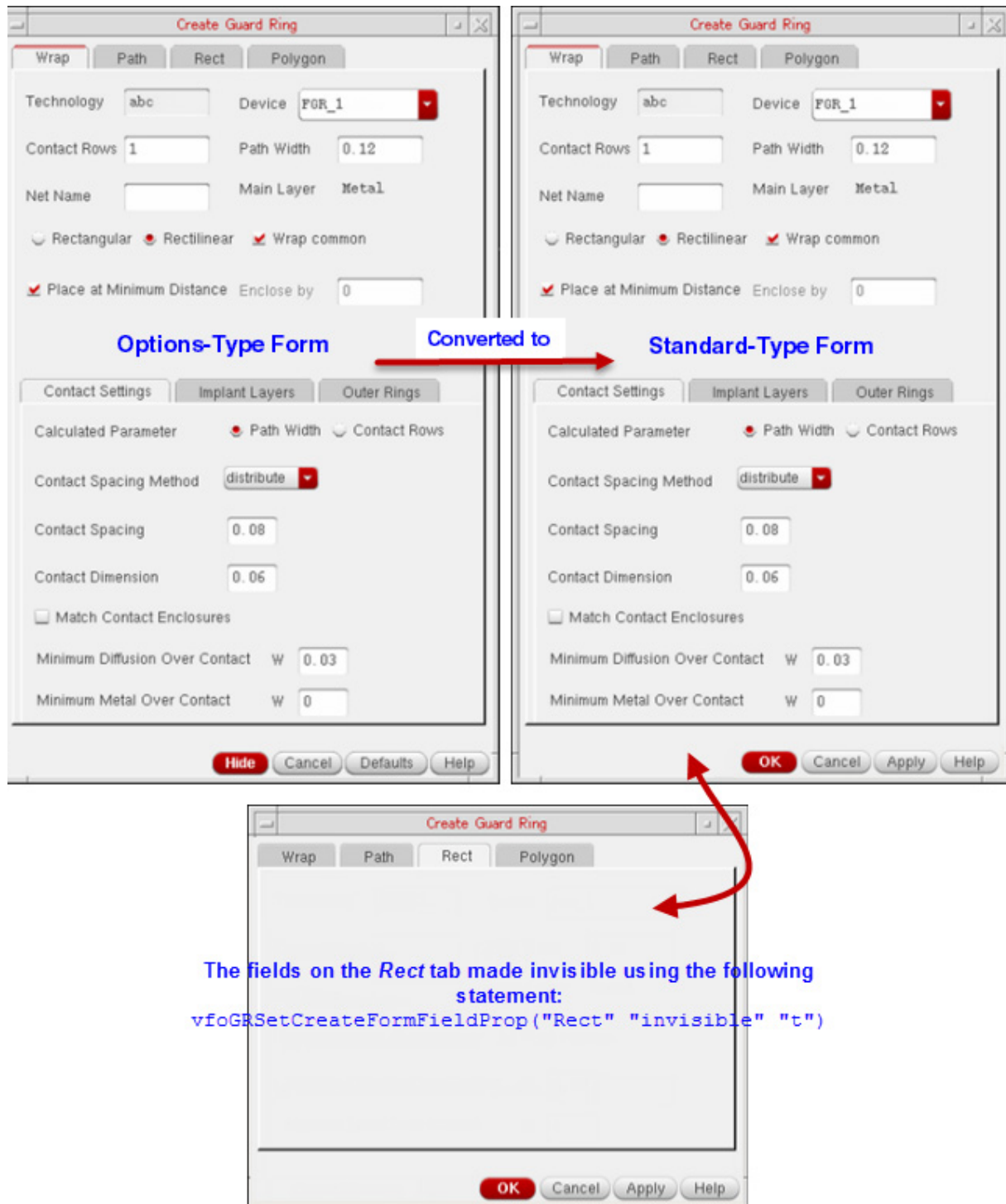
Customize Create Guard Ring Form

```
vfoGRRegCreateFormUpdateCallback (form_modgen "vfoCustomize_modgen")
; define the callback
procedure (vfoCustomize_modgen (formPointer
    ; write your own method body here related
    ; to Create Guard Ring form updates
)
hiDisplayForm (form_modgen)
```

Virtuoso Fluid Guard Ring Developer Guide

Customize Create Guard Ring Form

The following figure shows the visual impact of this customization code on the Create Guard Ring form:



For more information, refer to the [Customizing Create Guard Ring Form](#) application note available on the [Cadence Online Support](#) website.

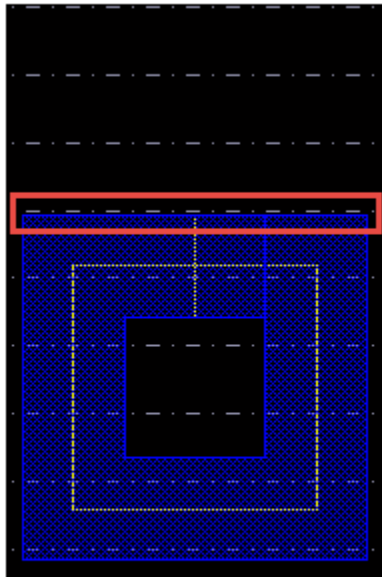
Using the Create Guard Ring Form

Use the Create Guard Ring form to specify the values to be considered while drawing an FGR instance on the layout canvas.

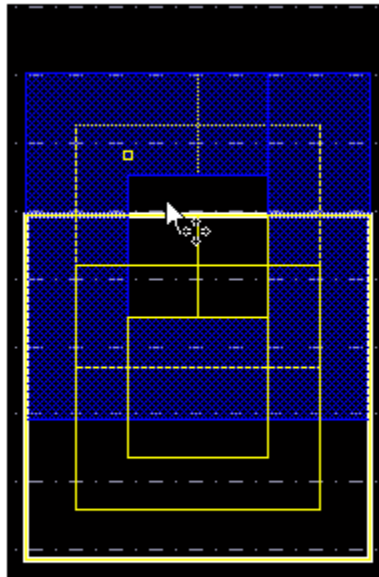
The VFO infrastructure supports automatic snapping of an FGR instance if snap pattern snapping is enabled in Layout L and relevant snap patterns are available on the canvas (just like snapping is available for any other instance in Layout L). When snapping is enabled, based on the snap pattern definitions, the FGR instance snaps to the closest snap pattern grid. For detailed information about snap pattern grids in Layout L, refer to the [FinFET Support in Layout L](#) chapter of the *Virtuoso Layout Suite L User Guide*.

The following images illustrate the difference in creating an FGR instance when automatic snapping is enabled or disabled in Layout L:

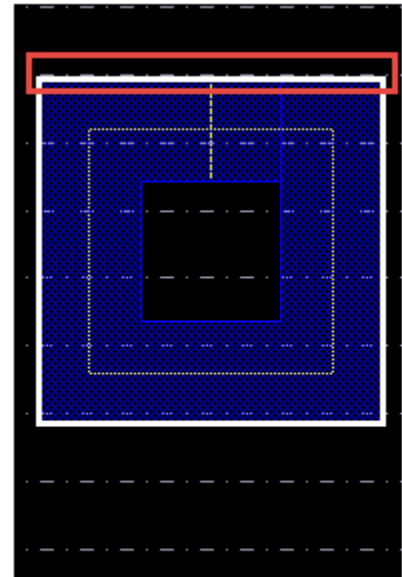
Automatic snapping to snap pattern grid is ***enabled***



1. When an FGR instance is created, it snaps to the closest snap pattern grid.



2. Drag the FGR instance to move it to a different snap pattern grid.

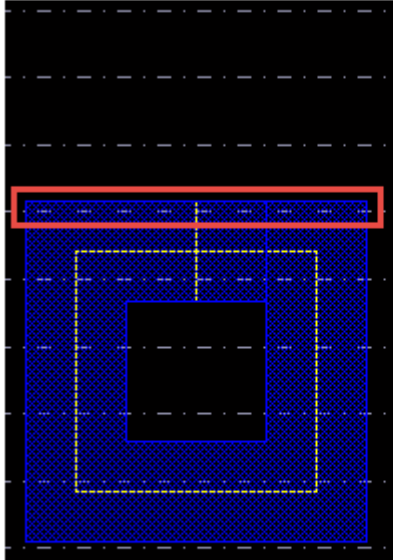


3. The FGR instance again snaps to the underlying snap pattern grid that is closest to the new position on the layout.

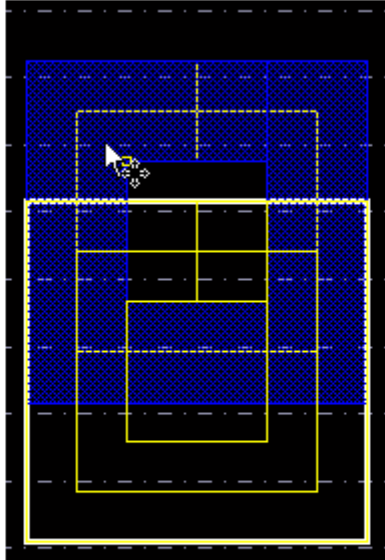
Virtuoso Fluid Guard Ring Developer Guide

Customize Create Guard Ring Form

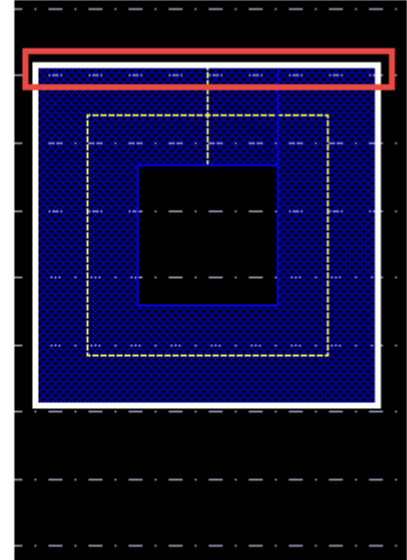
Automatic snapping to snap pattern grid is *disabled*



1. When an FGR instance is created, it does not snap to the closest snap pattern grid.



2. Drag the FGR instance to move it to a different snap pattern grid.



3. The FGR instance again does not snap to the underlying snap pattern grid that is closest to the new position on the layout.

Other SKILL Functions for Create Form Modifications

Triggers

vfoGRUpdateCreateForm(formPointer)

Changes the properties of the GUI components. A `formPointer` argument is available through this trigger to update the GUI components on the form.

Using this trigger, you can set the GUI components as visible or invisible, make those editable or non-editable, and set the default initial values of each field and much more.

Note: The `vfoGRUpdateCreateForm` trigger can be used to make all pre-defined GUI components except the *Technology* and *Device* fields invisible and add new GUI components as per requirement. To make all pre-defined, including the *Technology* and *Device* fields, and user-defined GUI components invisible, use the following SKILL function:

`vfoGRSetCreateFormFieldProp (promptName propertyName propertyValue)`

For possible values of the `promptName` argument of this SKILL function, see [Field Prompts in vfoGRSetCreateFormFieldProp SKILL Function](#).

vfoGRAddCreateFormFields()

Adds more GUI components to the Create Guard Ring form.

The `vfoGRAddCreateFormFields` trigger contains definition of all `hi*` components. This trigger is called every time the Create Guard Ring form is created.

Queues

vfoGRAddFieldsInCommonQ

Displays the specified user-defined GUI components on the Create Guard Ring form when the `vfoGRAddCreateFormFields` trigger is called. The required user-defined GUI components should be added to a list in the following format:

```
vfoGRAddFieldsInCommonQ=list(  
    list(<GUI_component>  
        <xPosition>:<yPosition>  
        <width>:<height>
```

Virtuoso Fluid Guard Ring Developer Guide

Customize Create Guard Ring Form

```
        [promptBoxWidthOrTitleHeight]
    )
)
```

Here,

- `xPosition` - Specify the upper-left `x` coordinate in pixels of the field relative to the upper-left corner of the form.
- `yPosition` - Specify the upper-left `y` coordinate in pixels of the field relative to the upper-left corner of the form.
- `width` - Specify the width of the field in pixels.
- `height` - Specify the height of the field in pixels.
- `promptBoxWidthOrTitleHeight` - Specify the width of the prompt box in pixels for fields that have a prompt and are laid out horizontally, such as a string or spin box field. However, in case of fields that have a top-down layout, such as the report field and the scroll region field, it specifies the height of the title box in pixels (the title is displayed at the top of the title box).

For example,

```
vfoGRAddFieldsInCommonQ=list(
    list(field1 10:50    180:20  120)
    list(field2 10:90    180:20  120)
    list(field3 10:130   180:20  120)
    list(chkbox1 10:210  180:20  160)
)
```

Field Prompts in vfoGRSetCreateFormFieldProp SKILL Function

You can use the field prompts given in the table below with the `promptName` argument of the `vfoGRSetCreateFormFieldProp` SKILL function. These help you to change the property of the specified field according to the requirement, such as, hiding it, making it editable or non-editable, assigning some value to it, and so on.

Field Description	Prompt Name
Create Guard Ring form	"Form"
Tabs of the form, that is, <i>Wrap</i> , <i>Path</i> , <i>Rect</i> , and <i>Polygon</i>	"Main Tabs"

Virtuoso Fluid Guard Ring Developer Guide

Customize Create Guard Ring Form

Field Description	Prompt Name
Wrap tab	"Wrap"
Path tab	"Path"
Rect tab	"Rect"
Polygon tab	"Polygon"
Technology field	"Technology"
Device field	"Device"
Main Layer field	"Main Layer"
Path Width field	"Path Width"
Contact Rows field	"Contact Rows"
Net Name field	"Net Name"
Rectangular option in Wrap tab	"Rectangular"
Rectilinear option in Wrap tab	"Rectilinear"
Wrap Common	"Wrap common"
Place at Minimum Distance	"Place at Minimum Distance"
Enclose By field	"Enclose by"
Snap Mode	"Snap Mode"
Automatically adjust to surround overlaps field	"Automatically adjust to surround overlaps"
Create Poly Fill field	"Create Poly Fill"
Create Poly Ring field	"Create Poly Ring"
Sub tabs i.e. Contact Setting, Implant Layers, and Outer Rings	"Sub Tabs"
Contact Settings tab	"Contact Settings"
Calculated Parameter field	"Calculated Parameter"
Contact Spacing Method field	"Contact Spacing Method"
Contact Spacing	"Contact Spacing"
Contact Dimension	"Contact Dimension"
Match Contact Enclosures field	"Match Contact Enclosures"

Virtuoso Fluid Guard Ring Developer Guide

Customize Create Guard Ring Form

Field Description	Prompt Name
Minimum Diffusion Over Contact label	"Minimum Diffusion Over Contact "
Minimum Diffusion Over Contact X field	"Minimum Diffusion Over Contact X"
Minimum Diffusion Over Contact Y field	"Minimum Diffusion Over Contact Y"
Minimum Metal Over Contact label	"Minimum Metal Over Contact "
Minimum Metal Over Contact X field	"Minimum Metal Over Contact X"
Minimum Metal Over Contact Y field	"Minimum Metal Over Contact Y"
Implant Layers tab	"Implant Layers"
Scroll field in Implant Layers tab	"Scroll Field"
Enclosure Cover Header label in Implant Layers tab	"Enclosure Cover Header Label"
Rows label under Outer Rings tab	"Rows Label "
Encl label under Outer Rings tab	"Encl Label "
Outer Rings tab	"Outer Rings "
Distance outer rings at minimum field under Outer Rings tab	"Distance outer rings at minimum"
Number of Rings	"Number of Rings"
Horizontal Pitch	"Horizontal Pitch"
Vertical Pitch	"Vertical Pitch"
Horizontal Segment Width	"Horizontal Segment Width"
Vertical Segment Width	"Vertical Segment Width"

Write Customized Fluid Editing Commands

A fluid shape of a fluid guard ring (FGR) device can be visually edited by a design editing tool or layout editor. In Layout L, a fluid shape is selectable from the top level. Therefore, it supports editing features like any other level 0 shape.

The different editing features are implemented through SKILL functions declared and defined in the VFO infrastructure. These functions provide the layout editor with the information required to know about and be able to edit a fluid shape. Therefore, when you edit a fluid shape, the corresponding set of SKILL updater functions are called and you can see editing on the layout canvas alongside your actions.

As a PDK developer, you have the flexibility to customize fluid editing commands. These commands are defined and controlled by a user-specified protocol class derived from the base protocol class. The following methods are available for you to specify the supported edit operations for an FGR (that is, enable or disable an edit command) and to redefine the behavior of edit operations.

Methods for Enabling or Disabling an Edit Command	Methods for Defining the Behavior of an Edit Command
<code>vfoSupportsChop?</code>	<code>vfoChopInstance</code>
<code>vfoSupportsMerge?</code>	<code>vfoMergeInstances</code>
<code>vfoSupportsCreateObstruction?</code>	<code>vfoCreateObstruction</code>
<code>vfoSupportsDeleteObstruction?</code>	<code>vfoDeleteObstruction</code>
<code>vfoSupportsConvertToPolygon?</code>	<code>vfoConvertToPolygon</code>
<code>vfoSupportsAbut?</code>	<code>vfoAbut</code>
<code>vfoSupportsUpdateModelShape?</code>	<code>vfoUpdateModelShape</code>

For syntactical details about these methods, refer to the section describing [vfoAdvAlgClass](#) in [Chapter 2, “Fluid Guard Ring Infrastructure.”](#) In case you are implementing fluid guard ring

without pitch support, refer to the section describing `vfoAbstractClass` class in [Chapter 2, “Fluid Guard Ring Infrastructure.”](#)

Defining Fluid Editing Commands

To write customized fluid editing commands, do the following:

■ In a SKILL code file:

- a. Define a user-defined protocol class that is extended from the base protocol class. For example, in the following code snippet, `sub1EditClass` has been extended from `vfoSfAdvImplClass`.

```
defclass( sub1EditClass (vfoSfAdvImplClass)
)
```

- b. Override methods for each required fluid editing feature and declare with it an object of the user-defined protocol class. For example, the following enables chop operation and defines the chop behavior by overriding `vfoSupportsChop?` and `vfoChopInstance` methods:

```
defmethod( vfoSupportsChop? ( ( obj sub1EditClass ) instId ) t
defmethod( vfoChopInstance ( ( obj sub1EditClass ) instId chopShapeData
@rest args )
prog( )
<define here the behavior of the chop command>
))
```

■ In the technology file:

- a. Declare your user-defined protocol class in the `tcDeclareDevice` construct, as shown below.

```
tcDeclareDevice( "layout" "cdsGuardRing" "NewDevice"
  ( (defComplementaryDevice "") (classVersion 1)
    (enclosureClass "vfoSfEnclosureClass")
    (vfoProtocolClass "sub1EditClass")
  )
```

- b. Set the user-defined protocol class as a property of `vfoProtocolClass` in the `tfcDefineDeviceProp` construct, as shown below.

```
tfcDefineDeviceProp(
; (viewName      deviceName      propName      propValue)
  (layout        NewDevice       vfoProtocolClass "sub1EditClass")
)
```

For more related details, refer to the [Creating Fluid SKILL Pcells](#) application note available on the [Cadence Online Support](#) website.

Pitch Handling Support for Editing Commands

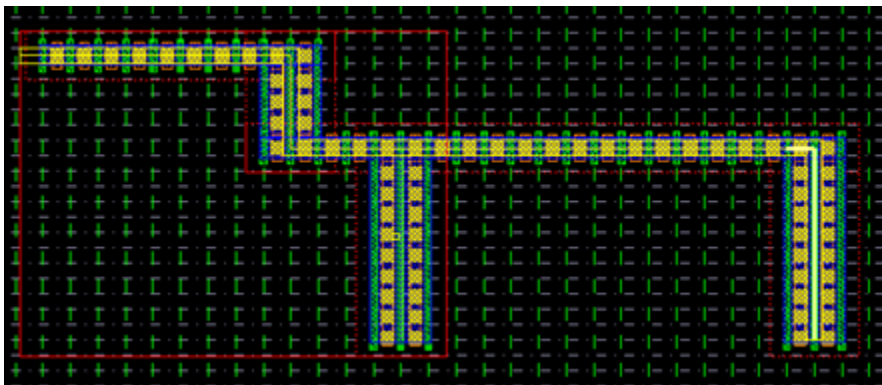
Stretch Command

For the *Stretch* command, pitch handling support is available only for orthogonal shapes, rectangle, path, and polygon, of `shapeType` "path".

Pitch handling is supported if the following options are selected in the Stretch form:

- *Snap Mode* option is set to *orthogonal*
- *Keep Guard Ring Ends Connected* option is selected

During the *Stretch* command, when you drag the mouse, the fluid shape snaps to the nearest available grid, as shown in the figure below.



Merge Command

The pitch handling support is available for the *Merge* command only if all of the following conditions are true:

- Ring-ring or path-path type shapes and `shapeType` is path
- Both instances have same net or either of them has a net
- The following parameters for both the instances are same for the same device type:
 - `horizontalPitch`

Virtuoso Fluid Guard Ring Developer Guide

Write Customized Fluid Editing Commands

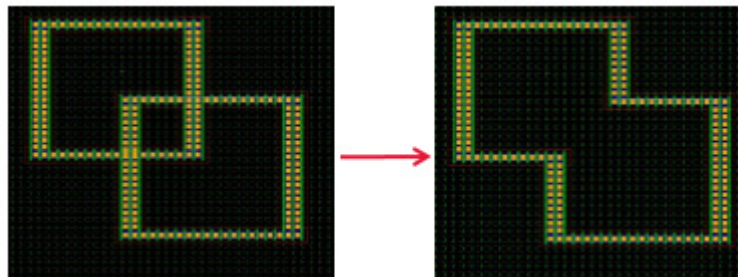
- ❑ verticalPitch
- ❑ horizontalSegWidth
- ❑ verticalSegWidth

You can use the `vfoGRCompareParams` SKILL API to check other parameters, as shown in the example below:

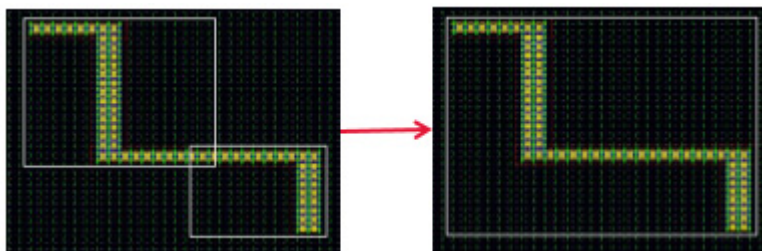
```
defmethod(vfoGRCompareParams ((obj userGuardRingClass) inst1 inst2)
  if(vfoGetParam(inst1 "userParam") == vfoGetParam(inst2 "userParam") t nil)
)
```

The results of the *Merge* command are illustrated below:

- During the ring-ring merge, for the merge to be successful, the `shapeData` of different fluid guard ring instances should overlap even if centerlines do not overlap. Also, the resulting shape is determined by the outermost edges of the fluid guard ring instances being merged.



- During path-path merge, the centerline of fluid shapes should be aligned at path ends that are to be merged. Also, the merge operation can be done only at path ends.



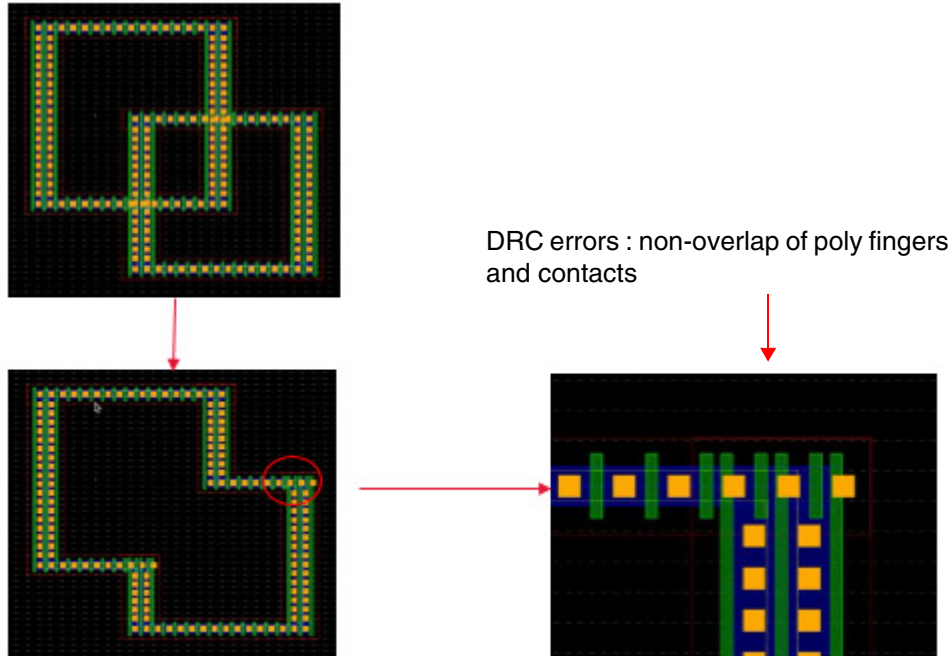
For cases where no global or local grids are available in the layout canvas, there is no common snapping reference for different fluid guard ring instances. In such cases, the fin grids and poly lines of different FGR instances do not align with each other while placing the instances. When such instances are merged, the resulting fluid guard ring segment lengths

Virtuoso Fluid Guard Ring Developer Guide

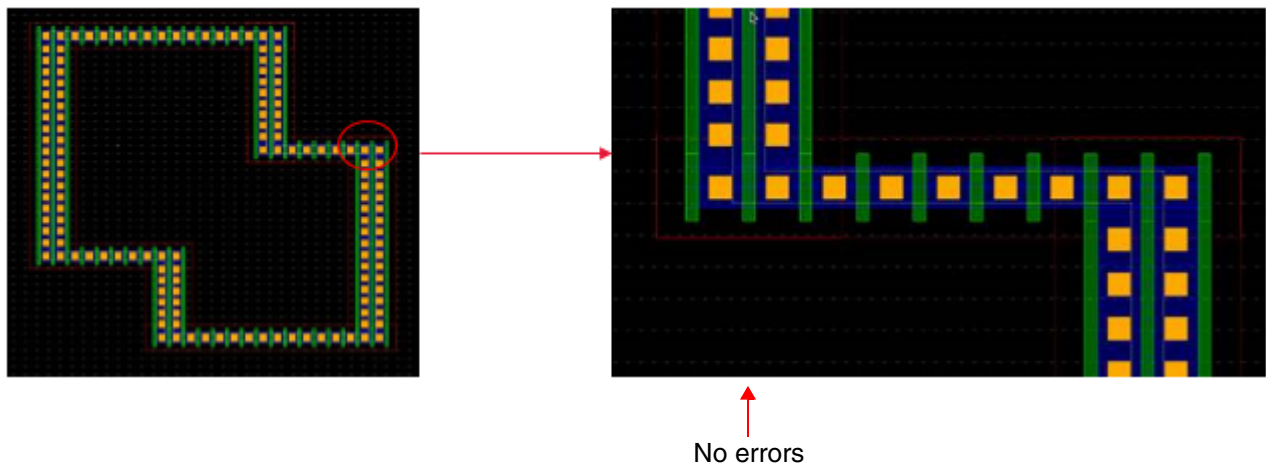
Write Customized Fluid Editing Commands

are not pitch correct and result in alignment and DRC errors. In such cases, the VFO infrastructure automatically performs post-edit pitch correction, as shown in the figure below:

Output without Post-edit Pitch Correction



Output with Post-edit Pitch Correction



However, the minimum segment length requirements and inner corner spacing, which result in DRC errors, are not taken care during the post-edit pitch correction. You should use the *Stretch* command to correct the segment lengths.

Note: You need to set the environment variable `fgrPostEditPitchCorrection` so that the VFO infrastructure post-processes the merged fluid guard ring instance to ensure that fluid shape data follows the pitch parameters.

Pitch Parameter Support in Edit Properties Form

The four pitch parameters, `horizontalPitch`, `verticalPitch`, `horizontalSegWidth` and `verticalSegWidth`, can be updated through the Edit Properties form. When you update these parameters, the shape data is also updated for the FGR instance.

Note: The change in the pitch parameters through the Edit Properties form or `vfoSetParam` function does not guarantee that the instance will snap to pitches.

Methodology to Maintain Versions of Implementation Class

You can maintain multiple versions of an implementation class. This is useful when you want to associate different drawing methods with an implementation class. Based on the implementation class version being used, different draw methods are called for creating the geometries in an FGR device. This chapter covers the methodology that VFO infrastructure provides to maintain versions of an implementation class.

For an FGR device, the technology file contains the following two parameters in the device definition:

- `formalVersion`
- `classVersion`

These parameters facilitate version control for an implementation class. Currently, the VFO infrastructure is implemented to support values 0 and 1 for these parameters.

A SKILL code file can be created to save the different version of an implementation class. In this file, the customized implementation class is extended from the `vfoGuardRing` class, as shown in the example below.

```
defclass(sublGuardRing (vfoAdvGuardRing)
(
    /* add here the new formal parameters in the guard ring definition */
) ;sublGuardRing
```

Different versions of this implementation class can be defined by sequentially extending one class from the other as shown in the figure below. Ensure that the class name string for each

Virtuoso Fluid Guard Ring Developer Guide

Methodology to Maintain Versions of Implementation Class

version is constructed based on the concatenation string present in the `vfoGuardRing` class definition in the technology file.

Technology File

```
tcCreateDeviceClass( "layout" "cdsGuardRing"
    ; class parameters
    ( (vfoGRImpl "vfoGuardRing") (modelLpp (quote ("y0" "drawing")))) ...
    ...
    (vfoGRGeometry
        (makeInstance
            (or
                (findClass
                    (concat vfoGRImpl "_ver_" formalVersion)
                )
                (_vfoMsg _vfoSCDoesNotExist117002
                    (concat vfoGRImpl "_ver_" formalVersion)
                )
            )
        )...
```

SKILL Code File

```
defclass(sub1GuardRing (vfoAdvGuardRing)
    (
        /* addition of new formal parameter in the guardRing definition */
    )
) ;sub1GuardRing
defclass(sub1GuardRing_ver_0 (sub1GuardRing) ());version control
defclass(sub1GuardRing_ver_1 (sub1GuardRing_ver_0);version control
    ()
) ;sub1GuardRing_ver_1
```

The device definitions for the customized implementation class, such as `sub1GuardRing` used in the example above, reside in the technology file. The figure below shows definition of device, `FGR_version1`, using the customized implementation class, `sub1GuardRing`, and

Virtuoso Fluid Guard Ring Developer Guide

Methodology to Maintain Versions of Implementation Class

parameter value, 1. Similarly another device definition, FGR_version0, can exist with implementation class as sub1GuardRing, but parameter value as 0 instead of 1.

Technology File

```
tcDeclareDevice( "layout" "cdsGuardRing" "NGR"
  ( (classVersion 1)
    (enclosureClass "vfoSfEnclosureClass")
    (vfoProtocolClass "vfoAdvSfImplClass")
    (highlightLpp (quote ("annotate" "drawing"))))
    (vfoGRImpl "sub1GuardRing")
    (mainLpp (quote ("Active" "drawing")))
    (modelLpp (quote ("y0" "drawing")))
    (tmpLpp (quote ("instance" "drawing")))
    (guardRingType "N")
    (termName "FGRTerm")
    (pinName "FGRPin")
    (defComplementaryDevice ""))
  )
  ( (shapeData "nil")
    (shapeType "none")
    (decompositionMode "fill45-path-poly")
    (hide_keepouts t)
    (fillStyle "distribute")
    (fillClass "vfoSfFillSafe")
    (debug 0)
    (do_something t)
    (formalVersion 1)
    (keepOuts (quote nil))
    (horizontalSegWidth 0.048)
    (verticalSegWidth 0.172)
    (verticalPitch 0.048)
    (horizontalPitch 0.086)
  )
)
```

In the SKILL code file, for each version of the implementation class, there should be a vfoSfDraw method that defines drawing of the geometries in an FGR instance. The example below shows the methods for sub1GuardRing_ver_0 and sub1GuardRing_ver_1:

```
defmethod(vfoSfDraw ((gr sub1GuardRing_ver_0))
  dbCreateRect(gr->cv list("Metal1") list(0:0 1:1))
```

Virtuoso Fluid Guard Ring Developer Guide

Methodology to Maintain Versions of Implementation Class

```
    ) ;vfoSfDraw
defmethod(vfoSfDraw ((gr sub1GuardRing_ver_1))
  dbCreateRect(gr->cv list("Metal2") list(1:1 2:2))
  ) ;vfoSfDraw
```

While creating an FGR device, these `vfoSfDraw` methods are evaluated based on `classVersion` and `formalVersion` parameter values present in device definition in the technology file.

Virtuoso Fluid Guard Ring Developer Guide

Methodology to Maintain Versions of Implementation Class

See the example below to understand this methodology.

SKILL Code File

```
defclass(sublGuardRing_ver_0 (sublGuardRing) ());version control
defclass(sublGuardRing_ver_1 (sublGuardRing_ver_0);version control
()
);sublGuardRing_ver_1
;body about generating of layout by writing the draw method
; will create a Rectangle shape on layout with Metall drawing layer
defmethod(vfoSfDraw ((gr sublGuardRing_ver_1))
  dbCreateRect(gr->cv list("Metall") list(0:0 1:1))
);vfoSfDraw
```

Customized
implementation class

Drawing method for
the customized
implementation class

Technology File

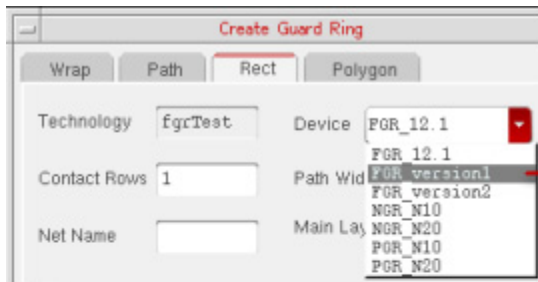
```
tcDeclareDevice( "layout" "cdsGuardRing" "NGR"
( (classVersion 1)
(enclosureClass "vfoSfEnclosureClass")
(vfoProtocolClass "vfoAdvSfImplClass")
(hilighLpp (quote ("annotate" "drawing")))
(vfoGRImpl "sublGuardRing")
(mainLpp (quote ("Active" "drawing")))
(modelLpp (quote ("y0" "drawing")))
(tmpLpp (quote ("instance" "drawing")))
(guardRingType "N")
(termName "FGRTerm")
(pinName "FGRPin")
(defComplementaryDevice "")
)
( (shapeData "nil")
(shapeType "none")
(decompositionMode "fill45-path-poly")
(hide_keepouts t)
(fillStyle "distribute")
(fillClass "vfoSfFillSafe")
(debug 0)
(do_something t)
(formalVersion 1)
(keepOuts (quote nil))
(horizontalSegWidth 0.048)
(verticalSegWidth 0.172)
(verticalPitch 0.048)
(horizontalPitch 0.086)
)
)
)
```

Definition of device,
FGR_version1, with
corresponding
implementation class,
sublGuardRing,
and version
parameter values are
set to 1.

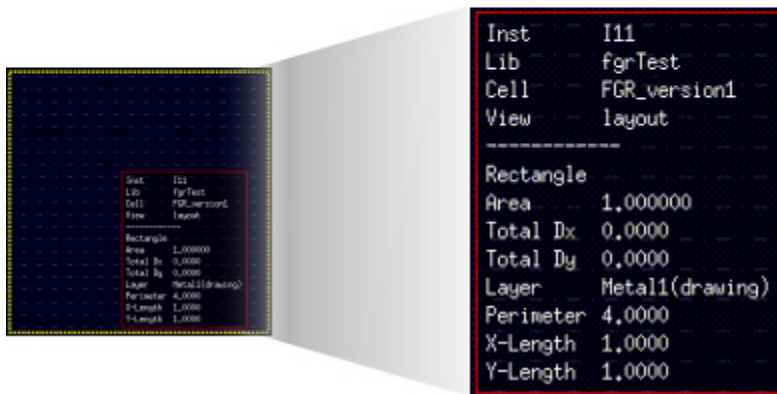
Device name as it will appear in the *Device* drop-down list box on Create Guard Ring form.

Virtuoso Fluid Guard Ring Developer Guide

Methodology to Maintain Versions of Implementation Class



Device name as defined in the `tfcDefineDeviceProp` method written in the `fgrTest` technology file.



The `vfoSfDraw` method for the `sub1GuardRing_ver_1` implementation class is called from the SKILL code and a Rectangle on *Metal1* drawing layer is drawn on the layout canvas, as shown on the left.

Fluid Guard Ring Packaging in PDK

The customized fluid guard rings (FGRs) are derived from the base classes defined in the Virtuoso Fluid Object (VFO) infrastructure. Therefore, the related SKILL and SKILL++ code files have a dependency on the `vfo*` context and class definitions. Typically, third-party tools cannot evaluate the customized FGRs because they do not have access to the SKILL and SKILL++ code written by a PDK developer specifically for such FGRs. Also, the approach of loading the SKILL and SKILL++ code from the `libInit.il` initialization file is not user intuitive. Therefore, to allow third-party tools to read customized FGRs, you need to load the `.il` and context files that have the VFO infrastructure definitions. This chapter explains how to load these files.

Note: A PDK developer handles the loading of files that contain the code for any customized FGR. Therefore, if you need assistance in configuring or troubleshooting the load sequence of customized FGRs, contact your PDK provider for information.

Run the following steps to manually load the VFO infrastructure:

1. Create a file that contains the lines of code given in the [Procedure for Initializing Customized FGR Devices](#) section and save it. For example,
`load_vfo_context_and_files.il`
2. Load this new SKILL file by adding the following lines of code in the `libInit.il` file:

```
load("load_vfo_context_and_files.il")
```

3. In the same `libInit.il` file, call the `load_vfo_context_and_files` SKILL procedure by using the following syntax:

```
load_vfo_context_and_files (cxt_path ils_path)
```

Here,

- `cxt_path` is the string specifying the location of the `vfo.cxt` file.

In the Virtuoso installation, the `cxt_path` is:

```
<install_dir>/tools/dfII/etc/context
```

- `ils_path` is the string specifying the location of the `vfo*.ils` files.

In the Virtuoso installation, the `ils_path` is:

`<install_dir>/tools/dfII/etc/vfo`

Procedure for Initializing Customized FGR Devices

Add the following procedure to a file, such as, `load_vfo_context_and_files.il`:

```
procedure(load_vfo_context_and_files(vfoCxtPath ilsPath)
  let((vfoLoadSeqFilePath fileName )

  unless(isContextLoaded("vfo")
    loadContext(sprintf(nil "%s/vfo.cxt" vfoCxtPath))
  );;unless

  vfoLoadSeqFilePath = strcat(ilsPath "/vfoInitialize.ils")

  if(isFileName(vfoLoadSeqFilePath)
  then
    when(!isCallable('vfoGRGeometry)
      load(vfoLoadSeqFilePath)
      foreach(fileName vfoGetFileListWithLoadSequence()
        load(sprintf( nil "%s/%s" ilsPath fileName))
        printf("done loading %s/%s\n" ilsPath fileName)
      )
    );;when

  else
    when(!isCallable('vfoGRGeometry)
      foreach(fileName (list
        "vfoMessageIds.ils"
        "vfoAbstractClass.ils"
        "vfoAddOns.ils"
        "vfoApi.ils"
        "vfoAlgClass.ils"
        "vfoUtils.ils"
        "vfoShapeData.ils"
        "vfoSfShapeData.ils"
        "vfoSf.ils"
        "vfoSfFilling.ils"
        "vfoGuardRing.ils"
        "vfoGrShrinkWrap.ils"
        "vfoGuardRingPreview.ils"))
```

Virtuoso Fluid Guard Ring Developer Guide

Fluid Guard Ring Packaging in PDK

```
    load(sprintf( nil "%s/%s" ilsPath fileName))
    printf("done loading %s/%s\n" ilsPath fileName)
  );;foreach
  );;when
  );;if
  );;let
);;load_vfo_context_and_files
```

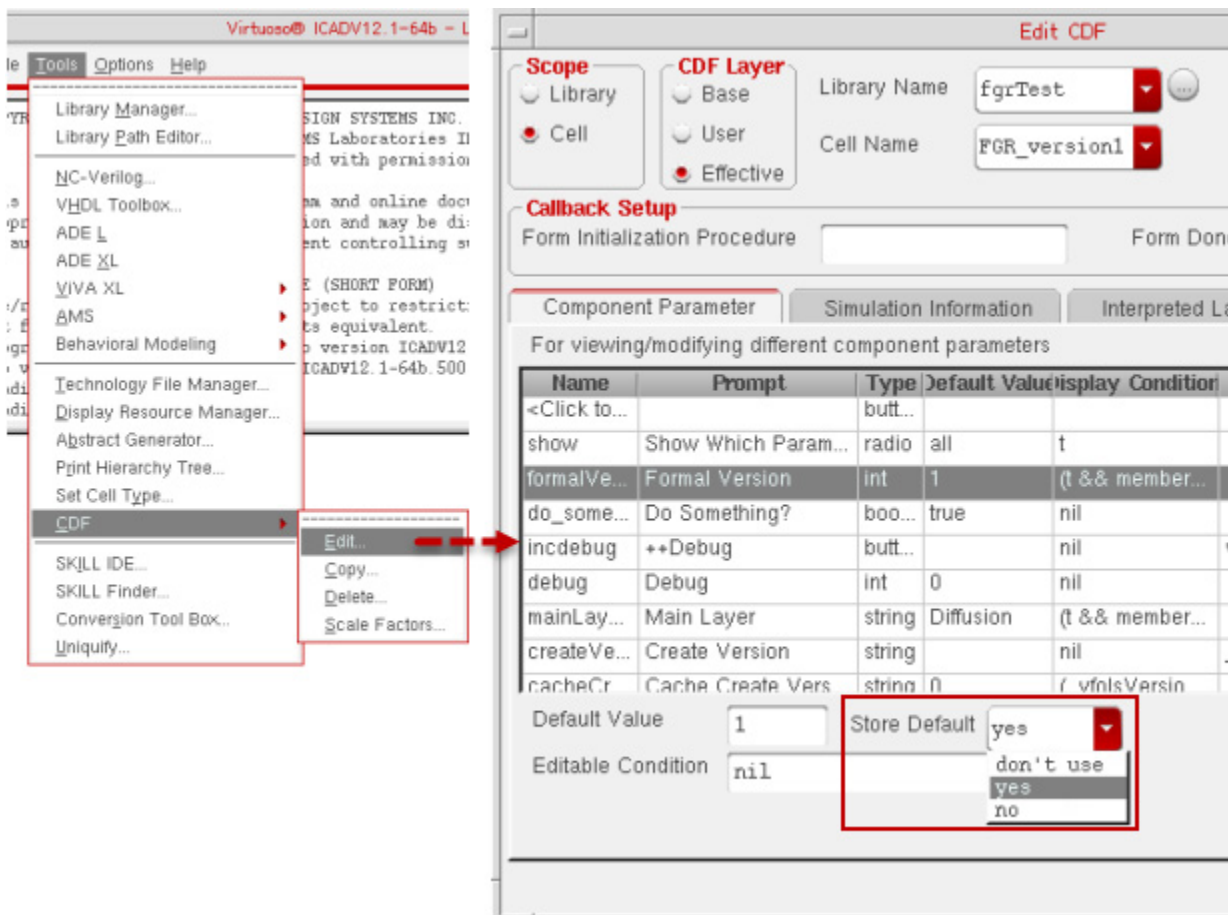
Virtuoso Fluid Guard Ring Developer Guide

Fluid Guard Ring Packaging in PDK

Best Practices for Developing a Fluid Guard Ring

While developing a fluid guard ring (FGR) device, Cadence recommends use of the following best practices to achieve reliable results:

- Enable store defaults, for example, by using the Edit CDF form. To access this form, in the CIW, choose *Tools – CDF – Edit*.



Virtuoso Fluid Guard Ring Developer Guide

Best Practices for Developing a Fluid Guard Ring

This concept is for all FGR parameters. If you want to retain the value on the instance even after the device definition or supermaster has been updated, the store defaults is used.

■ Run Cadence® SKILL Lint and Profiler assistant

SKILL Lint is useful in examining SKILL code for potential errors that went undetected during normal testing and ways to clean up your code. In particular, it helps programmers find unused local variables, global variables that should be locals, functions that have been passed the wrong number of arguments, and hints about how to improve the efficiency of their SKILL code. For detailed information, refer to the *SKILL Lint* appendix in *Cadence SKILL IDE User Guide*.

The Profiler assistant can be used to check the time and memory consumption of your SKILL programs. For detailed information, refer to the *Working with the Profiler Assistant* section of the *Examining Program Data* chapter in *Cadence SKILL IDE User Guide*.

- Do not use any three-letter lower case prefixes, such as `cdn`, for your function or procedure names. For detailed information, refer to the *Naming Conventions* section of the *Language Characteristics* chapter in *Cadence SKILL Language User Guide*.
- Avoid using private functions. Contact your Cadence representative when a public function is needed.
- To use extensibility of VFO infrastructure effectively:
 - ❑ Avoid using `callNextMethod()` because if the base class method implementation is changed, it can impact the sub-class functionality.
 - ❑ Always override the methods related to drawing or evaluation of FGR Pcell in the sub-class. For example: `vfoGRGeometry()` and `vfoSfDraw()`.
 - ❑ Ensure `vfoSfEnclosureClass` and `vfoSfFillSafe` classes are used only for drawing an instance of FGR device. Also, ensure that the methods of these classes are called only in the `vfoSfDraw` method.
 - ❑ Make all the new files created by PDK developer for extensibility a part of the customer PDK.
- Load the VFO infrastructure manually if a third-party tool is not able to evaluate your customized FGR. This is needed because third-party tools do not have access to the SKILL and SKILL++ code written by a PDK developer. For detailed information, refer to *Chapter 8, “Fluid Guard Ring Packaging in PDK.”*
- Avoid redefining the trigger available for updating the Create Guard Ring form fields. If the trigger is redefined, it overwrites its previous setting.

Virtuoso Fluid Guard Ring Developer Guide

Best Practices for Developing a Fluid Guard Ring

Virtuoso Fluid Guard Ring Developer Guide

Best Practices for Developing a Fluid Guard Ring

Fluid Guard Ring Environment Variables

This appendix covers the FGR-specific Layout environment variable names, descriptions, types, and values. For information about the other Virtuoso® Layout Suite L layout editor and graphics editor environment variables, refer to the *Environment Variables* appendix in the *Virtuoso Layout Suite L User Guide*.

The graphic environment variables control the characteristics of the window display and the layout environment variables control how various layout editor commands work. Many graphic environment variables have duplicate layout environment variables. In these cases, the layout variable supersedes the graphic variable unless the graphic variable is stored in the cellview. You can set both graphic and layout environment variables.

Setting Environment Variables

You can set the environment variables in the following three ways:

- Within the .cdsenv File
- Within the .cdsinit File
- In the CIW

.cdsenv File

Add environment variables to the `.cdsenv` file when the settings should be applied while launching Layout L.

Use the following syntax:

```
layout environmentVariableName dataType value
```

For example:

```
layout fgrWrapPlaceAtMinimumDistance boolean t
```

.cdsinit File

Like the `.cdsenv` file, the environment variable settings saved in the `.cdsinit` file get applied when you launch Layout L.

Use the `envSetVal()` command, which has the following syntax, to add environment variables to the `.cdsinit` file:

```
envSetVal("layout" "environmentVariableName" 'dataType value)
```

For example:

```
envSetVal("layout" "fgrWrapPlaceAtMinimumDistance" 'boolean t)
```



The `dataType` should be preceded with a single quote, else the command will not work. Also, if `dataType` is `string`, enclose its `value` within double quotes.

CIW

Use the `envSetVal()` command in the CIW to set an environment variable for the duration of the current session. The syntax is the same as described above for the `.cdsinit` file.

Note: Alternatively, to set environment variables for a single session, you can include the `envSetVal()` command in any Cadence SKILL file that you load.

If you use the CIW to set an environment variable that controls a widget on the currently open form, the implemented settings get reflected only after you close the form and then re-open it.

Displaying the Current Value of an Environment Variable

To determine the current value of any Layout L environment variable, use the following syntax in the CIW.

```
envGetVal("layout" "environmentVariableName")
```

List of Environment Variables

The following environment variables are available for use:

- fgrPostEditPitchCorrection
- fgrWrapPlaceAtMinimumDistance
- fluidGuardRingInstallPath
- grEnclosedBy
- grMode
- keepGuardRingEndsConnected
- vfoShowOnlyFluidShapeForDrag

fgrPostEditPitchCorrection

Syntax

In the `.cdsenv` file:

```
layout fgrPostEditPitchCorrection boolean { t | nil }
```

In the `.cdsinit` file or the CIW:

```
envSetVal("layout" "fgrPostEditPitchCorrection " 'boolean {t | nil})
```

Description

Controls whether the VFO infrastructure will post-process the merged fluid guard ring instance to ensure that fluid shape data follows the pitch parameters.

Default Value: `nil`

fgrWrapPlaceAtMinimumDistance

Syntax

In the `.cdsenv` file:

```
layout fgrWrapPlaceAtMinimumDistance boolean { t | nil }
```

In the `.cdsinit` file or the CIW:

```
envSetVal("layout" "fgrWrapPlaceAtMinimumDistance" 'boolean {t | nil})
```

Description

Controls the selection of the *Place at Minimum Distance* check box on the *Wrap* tab of the Create Guard Ring form.

When `fgrWrapPlaceAtMinimumDistance` is set to `t`, the *Place At Minimum Distance* check box is selected and the *Enclose by* field is disabled. However, setting the environment variable to `nil`, deselects the check box and makes the *Enclose by* field editable for specifying the guard ring distance from the object.



While using customized FGRs, setting the `fgrWrapPlaceAtMinimumDistance` environment variable to `nil` is recommended.

Default Value: `t`

Examples

- Deselect the *Place At Minimum Distance* check box and make the *Enclose by* field editable:

```
envSetVal("layout" "fgrWrapPlaceAtMinimumDistance" boolean nil)
```

- Return the current value of the `fgrWrapPlaceAtMinimumDistance` environment variable:

```
envGetVal("layout" "fgrWrapPlaceAtMinimumDistance")  
=> nil
```

fluidGuardRingInstallPath

Syntax

In the `.cdsenv` file:

```
layout fluidGuardRingInstallPath string alternatePath
```

Description

Enables you to specify the path from where the FGR-related SKILL files (`vfo*.ils`) are to be loaded. By default, the value string is empty, in which case these SKILL files are loaded from the following default release installation directory:

```
<install_dir>/tools/dfII/etc/vfo
```

Note: This method of specifying the path from where to load the FGR-related SKILL files (`vfo*.ils`) is not recommended for customized FGRs. If you want to use it, contact your Cadence® Customer Support representative.

Default Value: " " (null string)

Example

Load the FGR-related SKILL files from a path other than the default release installation directory:

```
layout fluidGuardRingInstallPath string /grid/cic/tool.lnx86/dfII/etc/vfo
```

grEnclosedBy

Syntax

In the `.cdsenv` file:

```
layout grEnclosedBy <any_positive_floating_point_number>
```

In the `.cdsinit` file or the CIW:

```
envSetVal("layout" "grEnclosedBy" <any_positive_floating_point_number>)
```

Description

Initializes the *Enclose by* field on the Create Guard Ring form.

You can replace the default value with another floating point number by updating this environment variable. Otherwise, change the value for the *Enclose by* field by typing in the field on the Create Guard Ring form.

Default Value: 0.0

grMode

Syntax

In the `.cdsenv` file:

```
layout grMode string { Rectangular | Rectilinear }
```

In the `.cdsinit` file or the CIW:

```
envSetVal("layout" "grMode" 'string { "Rectangular" | "Rectilinear" })
```

Description

Sets the default FGR creation type in *Wrap* mode.

By default, this environment variable is set to `Rectilinear`. Therefore, when you launch the Create Guard Ring form, on the *Wrap* tab, the `Rectilinear` radio button is selected.

However, when you reset this environment variable to `Rectangular`, the `Rectangular` radio button appears as selected on the *Wrap* tab.

This environment variable is useful when you mostly create `Rectangular` type of FGR in *Wrap* mode and want that be selected as the default creation type each time you access the Create Guard Ring form.

Default Value: `Rectilinear`

keepGuardRingEndsConnected

Syntax

In the `.cdsenv` file:

```
layout keepGuardRingEndsConnected boolean { t | nil }
```

In the `.cdsinit` file or the CIW:

```
envSetVal("layout" "keepGuardRingEndsConnected" 'boolean {t | nil})
```

Description

Controls whether a guard ring with touching ends will stay connected (`t`) or will result in opening up the guard ring (`nil`) during the *Stretch* and *Quick Align* command operations.

Default Value: `nil`

vfoShowOnlyFluidShapeForDrag

Syntax

In the `.cdsenv` file:

```
layout vfoShowOnlyFluidShapeForDrag boolean { t | nil }
```

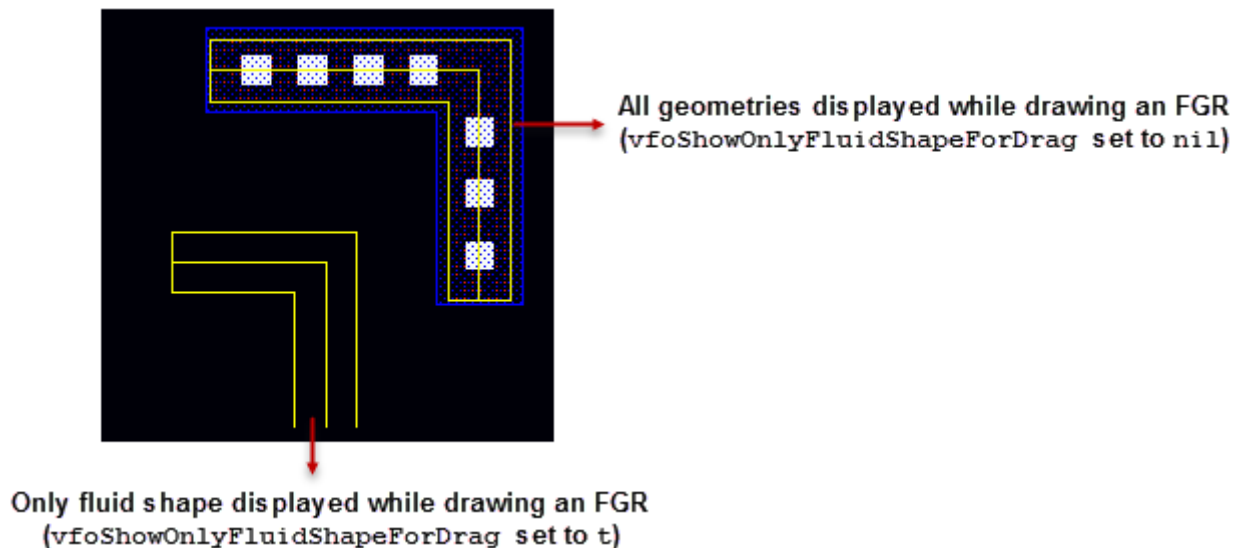
In the `.cdsinit` file or the CIW:

```
envSetVal("layout" "vfoShowOnlyFluidShapeForDrag" 'boolean {t | nil})
```

Description

Controls the display of the fluid shape and other geometries like metal layer, diffusion layer, and contacts while drawing an FGR on the layout canvas.

When `vfoShowOnlyFluidShapeForDrag` is set to `nil`, all geometries including the fluid shape are visible as you draw the FGR on the layout canvas. However, setting the environment variable to `t` displays only the fluid shape. The figure below shows the difference in the two approaches.



Default Value: `nil`

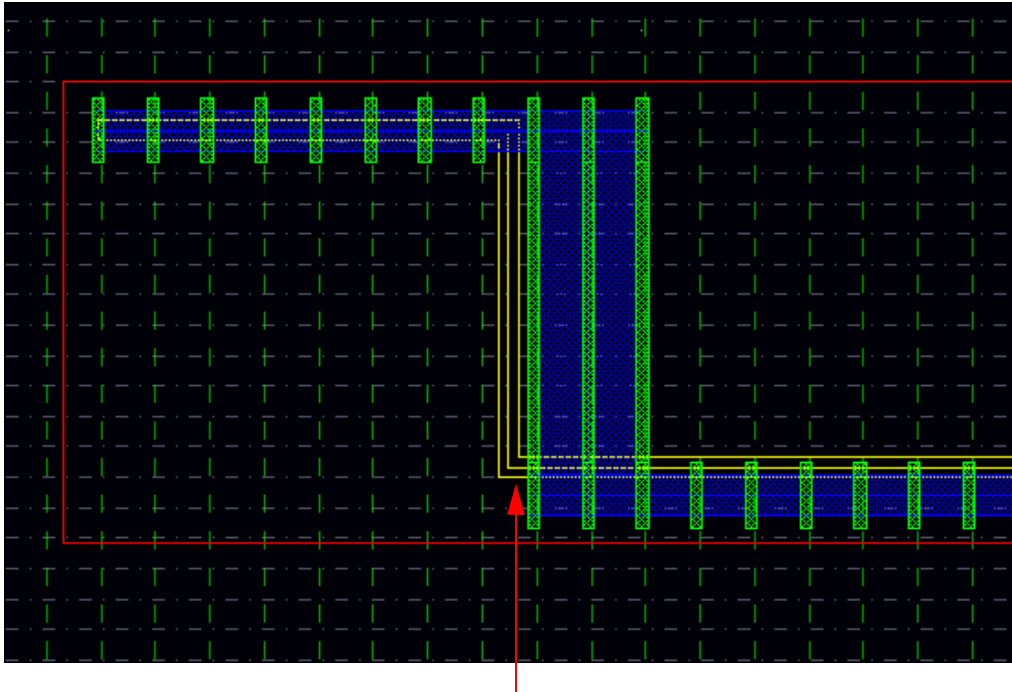
Fluid Guard Rings Known Problems and Solutions

This document describes the known issues with Virtuoso® Fluid Guard Ring (FGR) and suggests workarounds for these issues. The issues are identified by a Cadence Change Request (CCR) number, wherever applicable. Unless otherwise stated, the issues described in this document were identified in ICADV12.3 release.

Drag lines do not follow the pitch for both horizontal and vertical direction.

During interactive creation and editing of FGR instances, the drag lines do not coincide with the evaluated shape due to shape data adjustment based on pitch parameters.

CCR 1199564: Provide user registered callback function for snapping enter point of rubber-band (drag-set) line



Drag line not coinciding with the shape

Solution: This issue has been documented for your information only. There is currently no workaround available.

Orthogonal snap mode is supported

During interactive creation and editing of an FGR instance, only orthogonal snap mode is supported.

Solution: Use the SKILL function [leSetFormSnapMode](#) for snapping the mouse pointer in the orthogonal direction.

Pitch support has been added only for 'Path' type shapeData

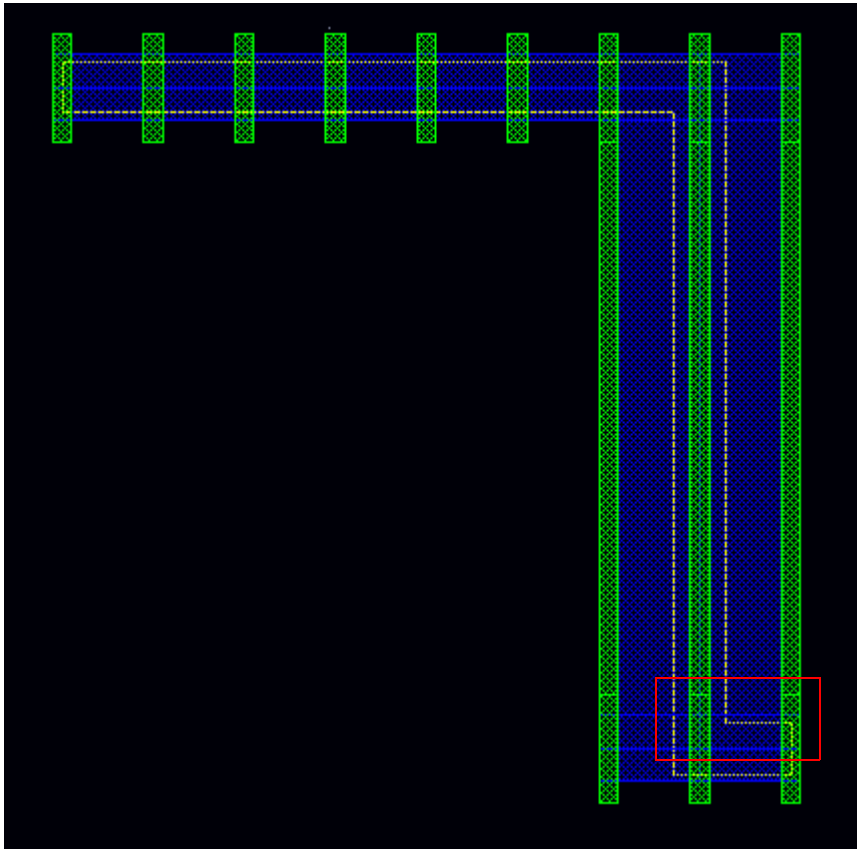
Pitch support classes cannot be used for rectangular and polygon type shapeData.

Solution: This issue has been documented for your information only. There is currently no workaround available.

Stretching an FGR sometimes results in DRC violation

Stretching an FGR can result in DRC violations when a horizontal/vertical segment length is less than half of the vertical/horizontal segment width.

CCR 1325427: Stretching an FGR sometimes creates contacts out of active area leading to DRC violation

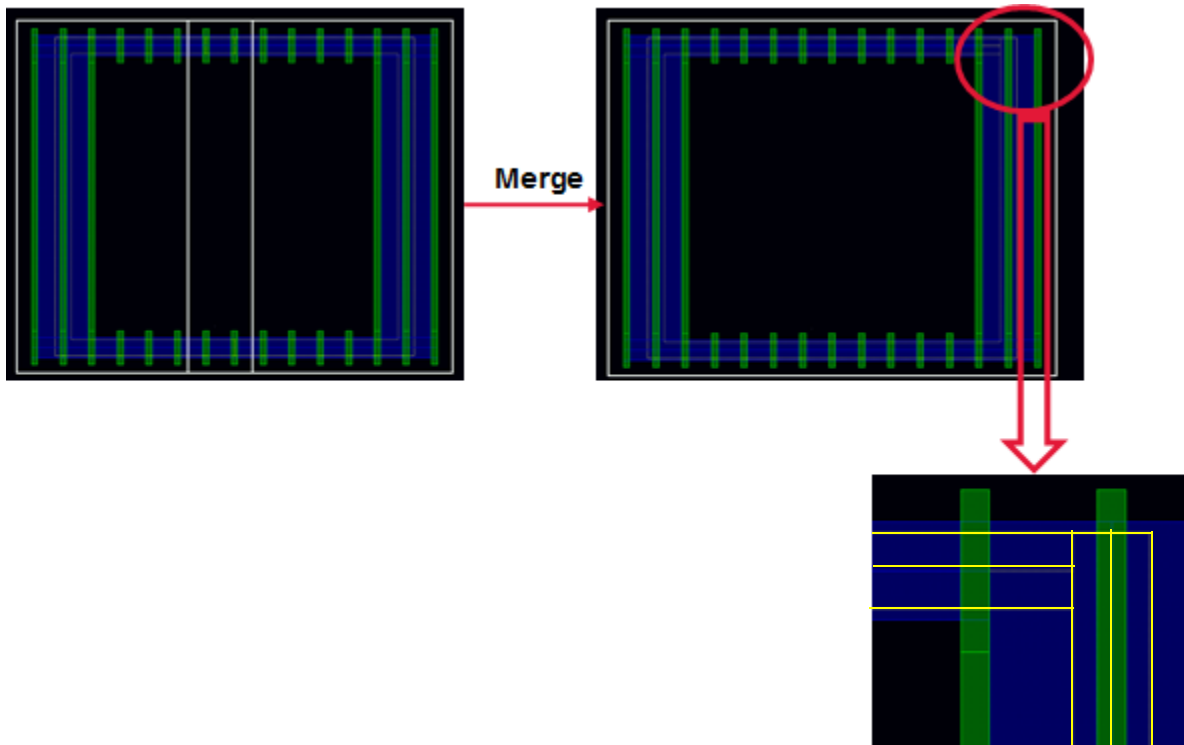


Horizontal segment length is less than half of the vertical segment width

Solution: This issue has been documented for your information only. There is currently no workaround available.

Ring structure is defied during the merge operation

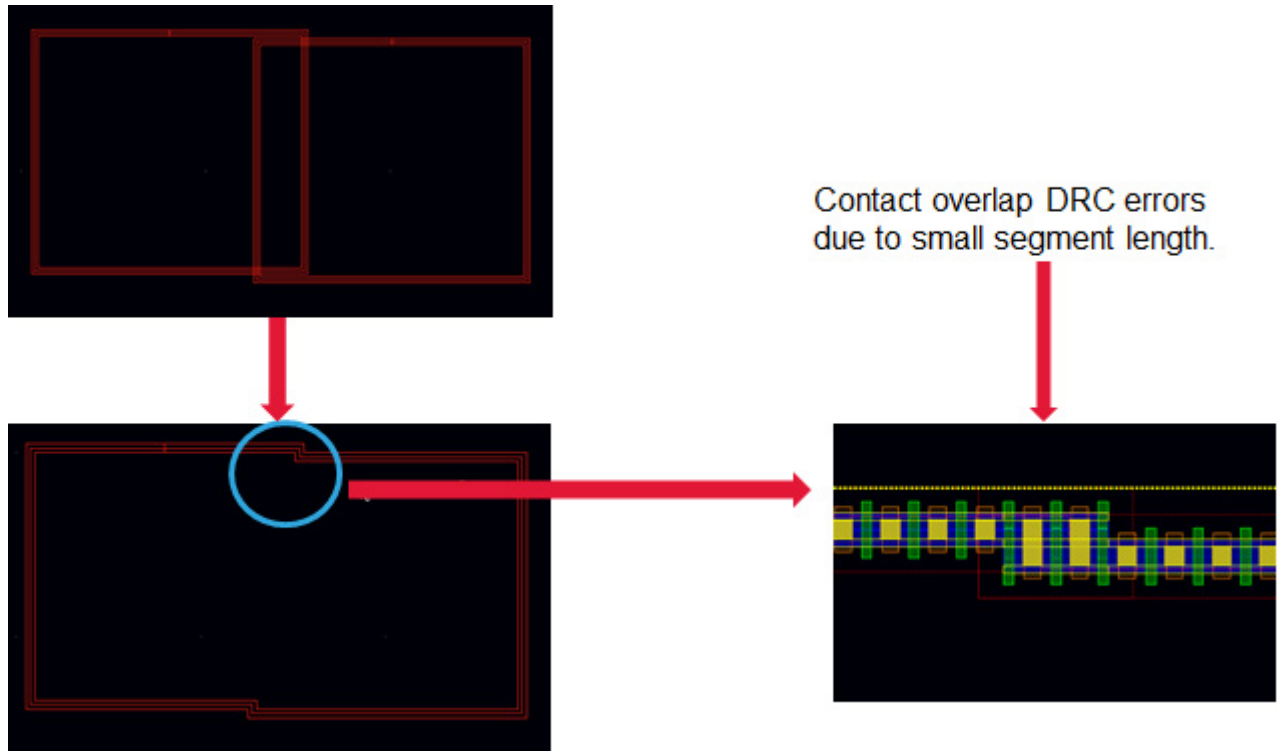
If two FGR instances are merged such that the resultant FGR is a ring type structure, the merged FGR is not created as a ring. This is because the centerline of the paths do not coincide, as shown in the figure below.



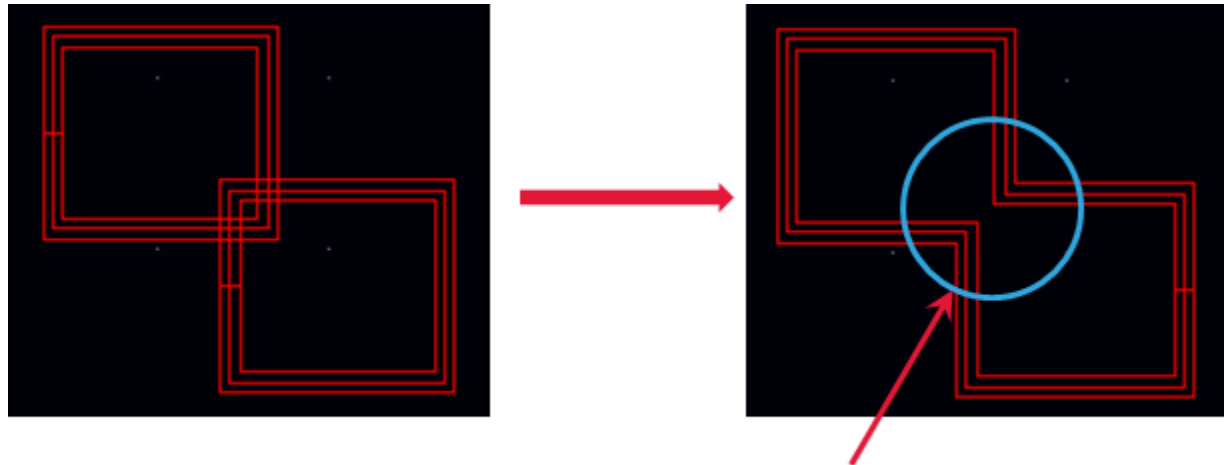
Solution: This issue has been documented for your information only. There is currently no workaround available.

During the Merge operation, minimum segment length and inner corner spacing can cause DRC errors.

Contact overlap DRC errors are caused because of small segment length, as shown in the figure below.



Contact overlap DRC errors are caused because of inner corner spacing, as shown in the figure below.



Inner corner spacing can lead to DRC errors

Solution: This issue has been documented for your information only. There is currently no solution available. As a workaround, you should use the Stretch command to increase the segment length or to correct the inner corner spacing.

Editing an FGR using the Edit Instance Property form is not supported.

Any instance parameters changed through the Edit Object Properties form does not cause re-evaluation of the FGR instance.

Solution: This issue has been documented for your information only. There is currently no workaround available.

The editing commands Reshape and Split do not support pitch parameters.

Solution: This issue has been documented for your information only. There is currently no workaround available.

Performance Improvement in Tunnel Command

This document describes the steps to migrate to the `vfoProtocolClass`, `vfoAdvSfImplEditClass`, to improve the performance of the `Tunnel` command in the ICADV12.3 release.

Migrating to `vfoAdvSfImplEditClass`

To migrate to `vfoProtocolClass`, `vfoAdvSfImplEditClass`, follow these steps:

1. Derive a class from `vfoAdvSfImplEditClass`.

```
defclass( vfoCustomEditClass (vfoAdvSfImplEditClass)      ( ))
```

2. Define the following methods to incorporate user-defined capabilities or features:

(i) `vfoSupportsCreateObstructions?` ;; This function should return `t` to enable improved performance of the `Tunnel` command.

(ii) `vfoCreateObstructions` ;; To define user-defined capabilities or features

```
defmethod( vfoSupportsCreateObstruction? (( obj vfoCustomEditClass)
instId) t)
```

```
defmethod( vfoCreateObstructions (( obj vfoCustomEditClass) inst
lppPointList @rest args)
    ;;user-defined capabilities or features
    ;;callNextMethod()
    ;;user-defined capabilities or features
    );;
```

3. Update the technology file for a custom FGRs as described below:

- a. Update the class parameter, `vfoProtocolClass` to `vfoCustomEditClass`

Virtuoso Fluid Guard Ring Developer Guide

Performance Improvement in Tunnel Command

b. Update the property vfoProtocolClass to vfoCustomEditClass

```
;;; device class "cdsGuardRing":  
:cDeclareDevice( "layout" "cdsGuardRing" "customFGR"  
  ( (classVersion 1) (enclosureClass "vfoSfEnclosureClass") (vfoProtocolClass "vfoCustomEditClass")  
    (highlightLpp (quote  
      ("annotate" "drawing")  
    )) (vfoGRImpl "vfoGuardRing") (mainLpp (quote  
      ("Oxide" "drawing")  
    ))  
  )  
)
```

```
tfcDefineDeviceProp(  
  (viewName deviceName  
    (layout customFGR  
      (propName propValue)  
      vfoProtocolClass "vfoCustomEditClass")  
    )  
  )  
)
```