

Virtuoso Parameterized Cell Reference

**Product Version IC23.1
September 2023**

© 2023 Cadence Design Systems, Inc.
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.

Cadence is committed to using respectful language in our code and communications. We are also active in the removal and replacement of inappropriate language from existing content. This product documentation may however contain material that is no longer considered appropriate but still reflects long-standing industry terminology. Such content will be addressed at a time when the related software can be updated without end-user impact.

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

1

<u>Overview</u>	11
<u>What Is a Parameterized Cell?</u>	13
<u>Pcell Super and Submaster Cells</u>	13
<u>Advantages of Using Pcells</u>	14
<u>Components of Pcell</u>	14
<u>How can Pcell Source be Represented</u>	15
<u>Using SKILL</u>	15
<u>Using SKILL++</u>	15
<u>Using Graphical Entry</u>	15

2

<u>Safety Rules for Creating SKILL Pcells</u>	17
<u>SKILL Procedures Called in Pcells</u>	18
<u>Macros in Pcell SKILL Code</u>	18
<u>Physical Limits for Functions</u>	19
<u>Recommended and Supported SKILL Functions for Pcells</u>	19
<u>Using Print Functions in a Pcell</u>	20
<u>Enclosing the Body of Code in a let or prog for Local Variables</u>	21
<u>What to Avoid When Creating Pcells</u>	21

3

<u>How to Package a Pcell</u>	23
<u>Loading of Pcell Support Files</u>	23
<u>Delivery of a Pcell Library</u>	24
<u>Specifying Source Code for Stream In</u>	25

4

<u>Creating SKILL Parameterized Cells</u>	27
<u>The pcDefinePCell Function</u>	27
<u>Using the pcCellView Variable</u>	28
<u>Using the SKILL Operator ~> with Pcells</u>	28
<u>Defining Parameter Values</u>	29
<u>Building Geometries</u>	31
<u>Creating Instances within Pcells</u>	32
<u>Building Nets, Terminals, and Pins</u>	32
<u>Accessing Technology File Data</u>	33
<u>Example 1</u>	34
<u>Example 2</u>	34
<u>Example 3</u>	34
<u>Variable Scoping</u>	34
<u>Schematic Symbol Pcells</u>	34
<u>Advantages of ROD Functions in Pcell</u>	35
<u>Using a Variable Number of Terminals</u>	36

5

<u>Creating SKILL++ Parameterized Cells</u>	39
<u>Advantages of SKILL++ Pcells</u>	39
<u>Recommended and Supported SKILL++ Functions for Pcells</u>	39
<u>Steps to Create SKILL++ Pcell</u>	40
<u>SKILL++ Pcell Samples</u>	41

6

<u>Creating Graphical Parameterized Cells</u>	47
<u>Adding the Pcell Menu</u>	48
<u>Stretch Commands</u>	48
<u>Drawing Stretch Control Lines</u>	49
<u>Defining Stretch Parameters</u>	52
<u>Specifying Stretch Directions</u>	52
<u>Specifying a Reference Dimension</u>	54
<u>Setting Minimum and Maximum Values</u>	55

Virtuoso Parameterized Cell Reference

<u>Applying Default, Minimum, and Maximum Values</u>	56
<u>Stretching Paths</u>	58
<u>Using Stretch with Repetition</u>	58
<u>Using Stretch with Conditional Inclusion</u>	63
<u>Stretch Menu</u>	64
<u>Conditional Inclusion Commands</u>	72
<u>Including or Excluding Objects</u>	72
<u>Using Conditional Stretch Control Lines</u>	73
<u>Using Conditional Inclusion with Repetition</u>	75
<u>Conditional Inclusion Menu</u>	76
<u>Repetition Commands</u>	81
<u>Specifying the Direction in which Objects Repeat</u>	82
<u>Specifying the Number of Repeated Objects</u>	84
<u>Specifying the Stepping Distance</u>	84
<u>Specifying Parameter Definitions</u>	85
<u>Repeating Pins and Terminals</u>	86
<u>Using Repetition with Stretch</u>	87
<u>Using Repetition with Conditional Inclusion</u>	93
<u>Repetition Menu</u>	93
<u>Parameterized Shapes Commands</u>	101
<u>Creating Parameterized Shapes</u>	102
<u>Defining a Margin</u>	102
<u>Parameterized Shapes Menu</u>	104
<u>Repetition Along Shape Commands</u>	108
<u>Using Control Path Segments</u>	108
<u>Specifying Start and End Offsets</u>	109
<u>Repetition Along Shape Menu</u>	112
<u>Reference Point Commands</u>	117
<u>Using a Reference Point Defined by Path Endpoint</u>	117
<u>Using a Reference Point Defined by Parameter</u>	120
<u>Reference Point Menu</u>	121
<u>Inherited Parameters Commands</u>	128
<u>Creating a Pcell with Inherited Parameters</u>	128
<u>Inherited Parameters Menu</u>	130
<u>Parameterized Layer Commands</u>	132
<u>Assigning a Parameterized Layer to Objects</u>	132

Virtuoso Parameterized Cell Reference

<u>Parameterized Layer Menu</u>	133
<u>Parameterized Label Commands</u>	136
<u>Assigning Parameterized Labels</u>	136
<u>Parameterized Label Menu</u>	137
<u>Parameterized Property Commands</u>	140
<u>Using Parameterized Properties</u>	140
<u>Parameterized Property Menu</u>	141
<u>Parameters Commands</u>	143
<u>Using Parameter Commands</u>	143
<u>Parameters Menu</u>	145
<u>Compile Commands</u>	149
<u>Using Compile</u>	149
<u>Creating a Pcell from a Cellview</u>	150
<u>Creating a SKILL File from a Cellview</u>	150
<u>Recognizing Error Conditions</u>	151
<u>Compile Menu</u>	151
<u>Recompile Pcells</u>	153
<u>Make Ultra Pcell Command</u>	154
<u>What Are Ultra Pcells?</u>	154
<u>Ultra Pcell Example</u>	154
<u>Ultra Pcell Requirements</u>	156
<u>Make Ultra Pcell</u>	157
<u>Customizing the Pcell Compiler</u>	160
<u>Background</u>	160
<u>How Customization Works</u>	160
<u>Variables</u>	161
<u>Restrictions on SKILL Functions</u>	162
<u>Example</u>	162
<u>Pcell Compiler Customization SKILL Functions</u>	163
<u>Creating Complex Pcells</u>	165
<u>Using Technology File Information</u>	165
<u>Specifying Values from the Technology File</u>	165
<u>Using Technology File Functions</u>	166
<u>Using the Component Description Format</u>	167
<u>Using Callbacks</u>	168
<u>Example of Pcell with CDF</u>	168

Virtuoso Parameterized Cell Reference

<u>Converting Graphical Pcells to SKILL Code</u>	171
<u>Examples of Converting Pcells to SKILL Code</u>	172

7

<u>Debugging SKILL Pcells</u>	179
<u>Working with Pcell IDE</u>	179
<u>Open the Pcell IDE Window</u>	179
<u>Pcell IDE Window Toolbar</u>	182
<u>Description</u>	183
<u>Debugging Pcell Supermaster</u>	185
<u>Debugging CDF</u>	191
<u>Debugging Pcell Instances</u>	192
<u>Publish Command</u>	199
<u>Debug Hier Mode Command</u>	201
<u>Pcell Evaluation Message</u>	202
<u>Supported and Not Supported Pcell Instances</u>	203
<u>Viewing Pcell Connectivity Model</u>	205
<u>Checking Pcell Parameters</u>	211
<u>Correcting and Updating Pcell Source</u>	213
<u>Restarting a Debugging Process</u>	213
<u>Debugging Pcells Abutment</u>	214
<u>Debug Abutment Form</u>	214
<u>Debugging Pcell Abutment using Pcell IDE</u>	216
<u>Check Abutment Command</u>	221
<u>Bindkey Editor</u>	224

8

<u>Express Pcells</u>	225
<u>Overview of Express Pcells</u>	226
<u>Express Pcell Plug-In for Non-Virtuoso and Third Party Applications</u>	226
<u>Requirements for Using the Express Pcell Feature</u>	227
<u>Virtuoso Design Environment Application</u>	227
<u>Non-Virtuoso Cadence Application</u>	227
<u>Third Party Application</u>	227
<u>Use Model of Express Pcell Management</u>	229

Virtuoso Parameterized Cell Reference

<u>Inside the Virtuoso Design Environment</u>	229
<u>Outside the Virtuoso Design Environment</u>	230
<u>Managing the Express Pcell Cache</u>	231
<u>Using Environment Variables</u>	231
<u>Using the Express Pcell Manager GUI in a Virtuoso Session</u>	234
<u>Maintaining the PDK Version in the Express Pcell Cache</u>	235
<u>Enabling Express Pcells for Specific Libraries</u>	237
<u>Installing the Express Pcell Plug-In for Non-Virtuoso or Third Party Applications</u>	240
<u>Cache Merge Utility</u>	241
<u>Preventing Data Inconsistencies</u>	241
<u>xpcmerge Command Line Utility</u>	242
<u>Sample Log File Format</u>	244
<u>Environment Variable to Control Express Pcell Proliferation Message</u>	245
<u>Exclude Specified Pcells from the Express Pcell Save or Read Operation</u>	246

A

<u>Graphical Pcells Form Descriptions</u>	249
<u>Compile To Pcell Form</u>	251
<u>Compile To Skill Form</u>	252
<u>Conditional Inclusion Form</u>	253
<u>Define Parameterized Label Form</u>	254
<u>Define Parameterized Layer Form</u>	256
<u>Define Parameterized Path Form</u>	258
<u>Define Parameterized Polygon Form</u>	259
<u>Define Parameterized Rectangle Form</u>	260
<u>Define/Modify Inherited Parameters Form</u>	261
<u>Delete Conditional Inclusion Form</u>	262
<u>Delete Parameterized Layer Form</u>	263
<u>Delete Parameterized Path Form</u>	264
<u>Delete Parameterized Polygon Form</u>	265
<u>Delete Parameterized Property Form</u>	266
<u>Delete Parameterized Rectangle Form</u>	267
<u>Delete Reference Point Form</u>	268
<u>Delete Reference Point by Path Form</u>	269
<u>Edit Parameters Form</u>	270

Virtuoso Parameterized Cell Reference

<u>Modify Conditional Inclusion Form</u>	271
<u>Modify Parameterized Label Form</u>	272
<u>Modify Parameterized Layer Form</u>	274
<u>Modify Repeat in X Form</u>	275
<u>Modify Repeat in X and Y Form</u>	276
<u>Modify Repeat in Y Form</u>	278
<u>Modify Repetition Along Shape Form</u>	279
<u>Parameterized Property Form</u>	280
<u>Reference Point by Path Endpoint Form</u>	281
<u>Reference Point by Parameter Form</u>	282
<u>Repeat in X Form</u>	283
<u>Repeat in X and Y Form</u>	284
<u>Repeat in Y Form</u>	286
<u>Repetition Along Shape Form</u>	287
<u>Stretch in X Form</u>	288
<u>Stretch in Y Form</u>	290
<u>Ultra Pcell Form</u>	292

B

<u>Express Pcell Manager</u>	295
------------------------------	-----

Virtuoso Parameterized Cell Reference

Overview

The Virtuoso® Parameterized cell software provides guidelines and Application Programming Interface (API) for creating Parameterized cells (Pcells) using SKILL and SKILL++. It also provides a user interface for creating graphical Pcells and debugging them. The more advanced Express Pcells infrastructure helps you evaluate SKILL Pcells in the Virtuoso design environment more efficiently, which can also be leveraged by non-Virtuoso Cadence and third-party applications.

This user guide describes how to create and debug Pcells. This user guide is meant for CAD engineers who are familiar with the development and design of integrated circuits, Virtuoso® Layout Suite L editor, and SKILL programming.

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

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

This chapter describes parameterized cells (Pcells) and the parameters assigned to them. The chapter discusses the following topics:

- What Is a Parameterized Cell?
 - Pcell Super and Submaster Cells
 - Advantages of Using Pcells
 - Components of Pcell
- How can Pcell Source be Represented
 - Using SKILL
 - Using SKILL++

Virtuoso Parameterized Cell Reference

Overview

- Using Graphical Entry

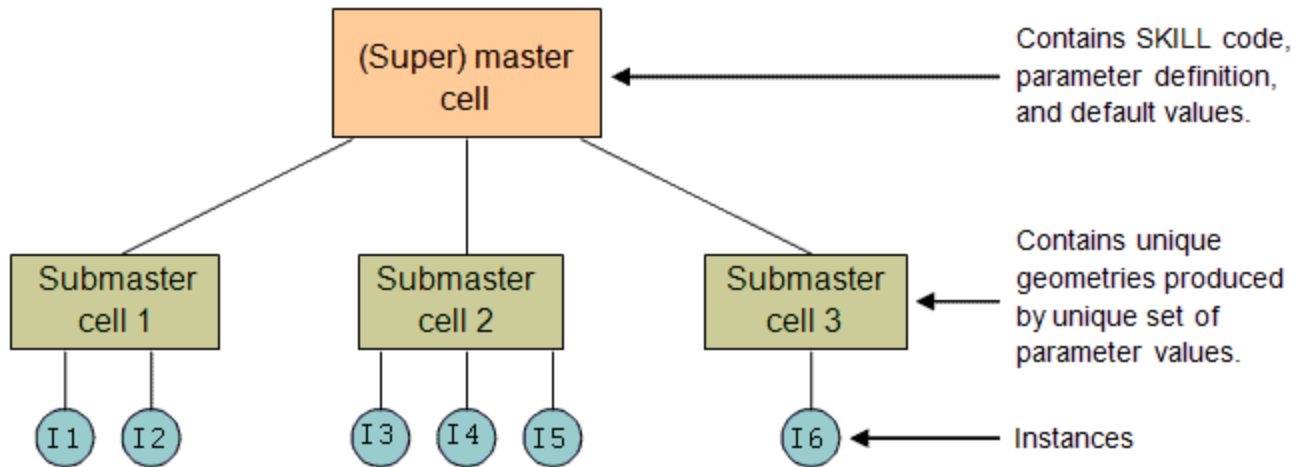
What Is a Parameterized Cell?

A parameterized cell, or Pcell, is a programmable cell that allows you to create a customized instance through a set of Pcell parameters. These parameters are used in the Pcell programming code. By providing different sets of values to these parameters, different sets of customized Pcells are created. A Pcell can be created either procedurally using SKILL functions or graphically using Virtuoso Pcell application.

Pcell Super and Submaster Cells

Each Pcell contains a supermaster and one or more submasters. A Pcell supermaster is created with the default values of the parameters and resided in hard disk. A Pcell submaster is created for each unique set of parameter values assigned to an instance of a supermaster Pcell and only exists in virtual memory.

See the figure below for the depiction of the Pcell architecture.



A submaster resides in virtual memory for the duration of your editing session and is accessible by all cellviews. When you create an instance in a cellview, and if a submaster with the same parameter values set already exists in virtual memory, a new submaster is not generated and the new instance references the existing submaster cell.

When you close all cellviews that reference a particular submaster cell, the database software automatically purges (removes) the referenced submaster cell from virtual memory but does not remove the master Pcell from your disk.

Virtuoso Parameterized Cell Reference

Overview

When you edit an editing session by exiting the Cadence software, the database software automatically purges (removes) all submaster cells from virtual memory. Purging does not remove master Pcells from your disk.

Advantages of Using Pcells

- Speed up entering layout data by eliminating the need to create duplicate versions of the same functional part.
- Save disk space by creating a library of cells for similar parts that are all linked to the same source.
- Eliminate errors that can result in maintaining multiple versions of the same cell.
- Eliminate the need to explore various levels of hierarchy when you want to change a small detail of a design.

Components of Pcell

A Pcell is made up of the following three components.

- Identification of Pcell Master

A Pcell master is identified by its library name, cell name, view name, and view type.

- Declaration of Pcell Parameters

The Pcell parameter is declared by its name, parameter type, and its default value. The parameter type is optional but recommended. It is necessary to specify the data type for Boolean parameters. If the default value is specified as 'nil' without specifying the data type, then the parameter value might be interpreted as the integer zero instead of the Boolean 'nil' (false).

It is common to use CDF parameters as Pcell parameters mainly for the parameter callback feature where the relationship between parameters can be established and validity of values can be enforced.

Using CDF enables you to add multiple values of a parameter. For example, you want to declare a parameter to control connection of gates and you have four choices: connect top pin of gates, connect bottom pin of gates, connect both pins of gates, and connect none pin of gates. Using CDF, you can declare the parameter with four cyclic choices. Within the Pcell code, you can use CDF APIs to read the current value of the parameter and do gate connections.

Virtuoso Parameterized Cell Reference

Overview

Note: When defining a CDF parameter, the default value must match the default value of a Pcell formal parameter.

■ Description of Pcell Contents

Pcell contents or source code uses SKILL/SKILL++ code for creating geometries and connectivity database objects. Enclose the code in a `let` or `prog` statement. If you use variables in the code, define them at the beginning of the `let` or `prog` statement. Using `let` gives faster performance than `prog`; `prog` allows multiple exits while `let` exits only at the end. It is recommended to call SKILL functions, supported application functions, and your own user-defined functions within the body of code.

How can Pcell Source be Represented

The identification and parameters of a Pcell are stored in the database in a persistent manner. The Pcell source can be completely or partially stored in the database and this is a trade-off between ease of delivery versus protection of intellectual property. For more information, see [How to Package a Pcell](#).

The Pcell source can be created manually. You can also create Pcell source automatically using the Graphical Pcell application.

Using SKILL

You can use SKILL function for the implementation of SKILL Pcell using the public SKILL function `pcDefinePCell`. For more information, refer to [Creating SKILL Parameterized Cells](#).

Using SKILL++

SKILL supports multiple programming paradigms, including object-oriented and functional programming styles. Therefore, it is relatively faster to write the integrated Pcell source code using the object oriented concepts with SKILL++ into the `pcDefinePCell` SKILL function. For more information, refer to [Creating SKILL++ Parameterized Cells](#).

Using Graphical Entry

The Virtuoso® parameterized cell software provides a graphical user interface that lets you create Pcells for placement in design layouts. Once you define the master cell and its

Virtuoso Parameterized Cell Reference

Overview

parameters with this tool, you can change parameter values for each instance you create in a layout cellview. For more information, refer to [Creating Graphical Parameterized Cells](#).

Safety Rules for Creating SKILL Pcells

This chapter provides important rules for creating SKILL Pcells, including which functions are safe to use and what to avoid.



If you use SKILL functions that are unsupported or not intended for use in Pcells, your Pcell code will probably fail when you try to translate your design to a format other than Design Framework II or when you use the Pcell in a different Cadence application.

The purpose for creating a Pcell is to *automate the creation of data*. Pcells should be designed as standalone entities, independent of the environment in which they are created and independent of the variety of environments in which you or someone else might want to use them. An environment can react to a Pcell, but Pcell code should not react to, interact with, or be dependent on an environment.

Although it is possible to create Pcells dependent on something in your current or local environment or using unsupported or not recommended functions, your Pcell code might fail on subsequent evaluations and is likely to fail when you try to translate it for a different environment. Although it is possible to load files, read from files, and write to files in the UNIX file system from within a Pcell, do not do so, because

- You cannot control the file permission settings in other locations, so reading or writing from a Pcell can cause the Pcell to fail in other directories, other environments, and during evaluation by translators.
- Loading a file from within a Pcell could cause the Pcell evaluation to fail. Depending on Pcell parameters, creating an instance might cause the system to execute the Pcell code again and reload the file. When a file is reloaded, the system issues a warning message that acts as output from the Pcell, causing the Pcell evaluation to fail.

Functions that are not supported for use by customers within SKILL Pcells usually belong to specific applications (tools); they are unknown to other environments, to other tools, and to data translators.

Virtuoso Parameterized Cell Reference

Safety Rules for Creating SKILL Pcells

Create Pcells using only the recommended, supported functions. You can identify them by their prefixes. However, you can also use all of the basic SKILL language functions that are defined in *Cadence SKILL Language Reference*; these functions do not have prefixes. To use only recommended, supported, documented functions, follow the rules in this section.

This chapter contains the following information:

- [SKILL Procedures Called in Pcells](#)
- [Macros in Pcell SKILL Code](#)
- [Physical Limits for Functions](#)
- [Recommended and Supported SKILL Functions for Pcells](#)
- [What to Avoid When Creating Pcells](#)
- [Using Print Functions in a Pcell](#)
- [What to Avoid When Creating Pcells](#)

SKILL Procedures Called in Pcells

You can call your own SKILL procedures from within Pcells. However, the code in procedures called by Pcells must follow all of the safety rules described in this section. The safety rules apply to all code that is evaluated when the system evaluates a Pcell, and that includes SKILL procedures called by the Pcell.

Called procedures are not compiled with the Pcell, so ensure they are loaded before the system evaluates the Pcell. You can set up the `libInit.il` file to automatically load the procedures the first time it accesses the library.

Related Topics

[How to Package a Pcell](#)

[Loading of Pcell Support Files](#)

Macros in Pcell SKILL Code

Macros in Pcells are only expanded one level. You can include macros that do not reference or contain other macros (are not nested) in your Pcell SKILL code, and they will be expanded successfully.

Virtuoso Parameterized Cell Reference

Safety Rules for Creating SKILL Pcells

To include a nested macro in a Pcell, it is recommended that you use an explicit function call in the Pcell code section of the `pcDefinePCell` function, and make sure that the function is defined and available (loaded) prior to any evaluation of the Pcell. This is usually done by including the function in your `libInit.il` file. If you do not use this method for nested macros, the nested code is treated as an undefined function, causing the Pcell evaluation to fail.

Physical Limits for Functions

You can avoid creating a Pcell that is too large to be processed or displayed by not exceeding the physical limitations for SKILL functions.

The following physical limitations exist for functions:

- Total number of *required* arguments is less than 65536
- Total number of *keyword/optional* arguments is less than 255
- Total number of local variables in a `let` is less than 65536
- Total number of local variables in a `let` must be less than 65536
- Max size of code vector is less than 1GB

By default, code vectors are limited to functions that can compile less than 32KB words. This translates roughly into a limit of 20000 lines of SKILL code per function. The maximum number of arguments limit of 32KB is mostly applicable in the case when functions are defined to take an *@rest* argument or in the case of *apply* called on an argument list longer than 32KB elements.

Recommended and Supported SKILL Functions for Pcells

When you create SKILL routines within Pcells, use only the following functions:

- The SKILL functions documented in *Cadence SKILL Language Reference*; for example, `car`, `if`, `foreach`, `sprintf`, `while`.
- SKILL functions from the `db`, `dd`, `cdf`, `rod`, `tech`, `abe`, `cst`, and `tx` families.
- You can use only SKILL functions, `pcExprToString`, `pcFix`, `pcRound`, and `pcTechFile` because:
 - Most supported `pc` functions correspond to the graphical user interface commands. You can use them to create Pcells in the graphical Pcell environment, but you cannot use them in the body of SKILL Pcell code.

Virtuoso Parameterized Cell Reference

Safety Rules for Creating SKILL Pcells

- ❑ Both the Pcell graphical user interface and the Pcell compiler are coded using `pc` SKILL functions. You cannot use the graphical user interface or compiler functions at all.

Related Topics

[Cadence SKILL Language Reference](#);

[pc Functions for SKILL Pcell Code](#)

Using Print Functions in a Pcell

You cannot print or generate output from a Pcell because it causes the Pcell evaluation to fail. However, you can bypass the system processing for print functions by using the `fprintf` or `println` function to print directly to `stdout` (standard output) in the format shown below:

```
fprintf( stdout ... )
println( variable stdout )
```

where *variable* is your variable name. For example

```
fprintf( stdout "myVariable = %L \n" myVariable )
```

Using printf or println in a Pcell

If you use the `printf` or `println` function in a Pcell, the output is treated as an error. In a cellview, errors are indicated by flashing markers. You can query the error in the cellview, change your Pcell code to fix the error, and recompile your Pcell.

To query markers,

1. Make the marker layer a valid layer.
2. Choose *Verify – Markers – Explain*.
3. Click a flashing marker.

A dialog box appears, showing the text printed from the Pcell by the `printf` or `println` function.

For debugging tips, see [Debugging SKILL Pcells](#).

Enclosing the Body of Code in a `let` or `prog` for Local Variables

When you use local variables in the body of SKILL Pcell code in a `pcDefinePCell` statement, be sure to enclose the Pcell code in a `let` or `prog` statement, and define all variables used in the Pcell code at the beginning of the `let` or `prog` statement. Defining variables as part of a `let` or `prog` prevents conflicts with variables used by the Pcell compiler. Using `let` gives faster performance than `prog`; `prog` allows multiple exits while `let` exits only at its end.

What to Avoid When Creating Pcells

When creating a Pcell, do not use functions with prefixes other than `db`, `dd`, `cdf`, `rod`, and `tech`. Although it is possible to call procedures with other prefixes from within a Pcell, you will not be able to view the Pcell in a different environment or translate it into the format required by another database. Most translators, including the physical design translators, can translate only basic SKILL functions, the four `pc` functions, and functions in the SKILL families `db`, `dd`, `cdf`, `rod`, and `tech`.

Note: In some cases, a function looks like a basic function but is not a basic function. For example, the Cadence analog design environment function `complex` looks like a basic function, but it is not. Avoid using Cadence analog design environment functions and other non-basic SKILL functions because the translators do not support them.

Specifically, **do not** use functions with application-specific prefixes such as `ael`, `abs`, `de`, `ge`, `hi`, `las`, `le`, `pr`, and `sch`. Procedures in any application-specific family are at a higher level of functionality than can be used in a Pcell safely. For example, if you create a Pcell using `le` functions, the Pcell works only in environments that include the Virtuoso layout editor. You cannot use a translator to export your design from the DFII database.

When a translator cannot evaluate a function, one of the following happens:

- The translator fails and issues an `undefined function` error message.
- The translator continues, issues a warning message, and translates the data incorrectly.

To make your design translate successfully, you must resolve undefined functions in Pcells. You can do this by flattening the Pcells, which results in a loss of hierarchy and parameterization, or by rewriting the Pcell code to eliminate the undefined functions.



Here are more important rules for creating Pcells. In your Pcell code,

- ❑ Do not prompt the user for input

Virtuoso Parameterized Cell Reference

Safety Rules for Creating SKILL Pcells

- ❑ Do not generate messages; message output is interpreted as an error
- ❑ Do not load, read, or write to files in the UNIX file system
- ❑ Do not run any external program that starts another process
- ❑ If you need to drive external programs to calculate cell shapes, do it in SKILL

Use CDF callback procedures to save the resulting list of coordinate pairs in a string, and then pass the string as input to a SKILL Pcell.

This method has the advantage that the external program needs to be called only once per instance, not each time the design is opened. For more information about callback procedures, refer to [Using the Component Description Format](#).

- ❑ Do not generate output with print functions, except as described in [Using Print Functions in a Pcell](#).

How to Package a Pcell

Pcell Library is a cellview library that contains Pcell supermaster cellviews and their Pcell Support Files. It is recommended to deliver Pcell Support File as context or encrypted files to protect the Pcell IP. For instance, Pcell as in supermaster cellviews and their Pcell Support Files are one components of a PDK.

When developing a Pcell, the developers need to provide the Pcell Definition File and Pcell Support Files. The Pcell Definition File contains the Pcell creation code that calls the function `pcDefinePCell` to create the on-disk Pcell (supermaster) cellview in the database. Pcell Support Files are the user-defined SKILL functions mentioned below:

- Functions used to support successful evaluation of Pcell
- Functions supporting CDF parameter callback
- Functions supporting Pcell abutment

These functions are usually put in separate files.



Important

Load all Pcell Support Files prior to referencing the Pcell in a design to avoid evaluation failure of Pcell, CDF callback, or Pcell abutment callback.

Loading of Pcell Support Files

DFII supports a mechanism to automatically load a SKILL file `libInit.il`, when a library is first accessed. The `libInit.il` file is written by Pcell developers to load the Pcell Support Files. If the Pcell Definition File is written in the top-level form, it should not be part of `libInit.il`. Each Pcell library should have one `libInit.il` file in the library directory. Pcell developers can organize the support files as required. One common example is to put all of the user SKILL functions in one SKILL directory. Then in each `libInit.il`, it loads the desired files from the SKILL location. Once each of the design libraries is opened, their `libInit.il` file and all the needed user SKILL functions will be loaded. The same approach can be applied to the ITDB libraries hierarchy. In each library level, it will only load its own

Virtuoso Parameterized Cell Reference

How to Package a Pcell

support files in its `libInit.il`. By opening the top-level library, all the referenced libraries will also be loaded automatically.

During development, Pcell developers can choose to load the Pcell Support Files as individual files or as SKILL context files consisting of these files. Loading as individual files allow the functions defined in them to be debugged with SKILL IDE or Pcell IDE.

For example:

```
if( status(debugMode) then
    load( file )
else
    loadContext( context )
)
```

Prior to the release of Pcell library, such a switch should be removed because only the protected files (context or encrypted) will be included in the release.

Note: Do not use the `.cdsinit` file to initialize Pcell code.

The `.cdsinit` is the initialization file used for interactive DFI workbenches. It is an environmental initialization or setup file, not a general data initialization file. The `libInit.il` file is supported at the database level and should be used for all data initialization purposes, such as Pcells and CDF called functions.

Delivery of a Pcell Library

If you create complex multiple Pcells, you can write your own utility SKILL procedures to call from the Pcells. Calling your own procedures from within a Pcell makes debugging the Pcell code easier. It can also provide better protection of your intellectual property when these procedures are placed in separate files from the file containing the call to `pcDefinePCell`.

What to Avoid in the libInit.il File

Do not load SKILL code that is dependent on graphical SKILL functions, such as `hiSetBindKey`, from your `libInit.il`. Assume that only low-level SKILL functions are available, such as those permitted in parameterized cells. When you use SKILL routines within `libInit.il`, use only the following functions:

- The SKILL functions documented in *SKILL Language Reference Manual*; for example: `car`, `if`, `foreach`, `sprintf`, `while`.
- SKILL functions from the following families: `db`, `dd`, `cdf`, `rod`, `tech`, `cst`.

Specifying Source Code for Stream In

When `libInit.il` is not provided in the data library, you must specify the source code files in the *User-Defined Skill File* field of the Stream In User-Defined Data form. For more information about this form and about the Design Data Translators, if you are using the OpenAccess database, refer to *Design Data Translators Reference*.

Virtuoso Parameterized Cell Reference

How to Package a Pcell

Creating SKILL Parameterized Cells

You can create Pcells in an ASCII file using SKILL instead of by using the graphical user interface. Complex Pcells are easier to create and maintain with SKILL than with the graphical interface.

Note: Cadence recommends that you consider using Virtuoso Relative Object Design (ROD) functions in your Pcell code. ROD functions help you in defining layout connectivity, geometries, and the relationships between them. You can make a Pcell stretchable using ROD. For information about ROD, see *[Virtuoso Relative Object Design User Guide](#)*.

This chapter contains the following information:

- [The pcDefinePCell Function](#)
- [Using the pcCellView Variable](#)
- [Using the SKILL Operator ~> with Pcells](#)
- [Defining Parameter Values](#)
- [Building Geometries](#)
- [Creating Instances within Pcells](#)
- [Building Nets, Terminals, and Pins](#)
- [Accessing Technology File Data](#)
- [Variable Scoping](#)
- [Schematic Symbol Pcells](#)
- [Advantages of ROD Functions in Pcell](#)

The pcDefinePCell Function

The `pcDefinePCell` function lets you pass a SKILL definition for a Pcell to the Pcell compiler. This gives you another way to create Pcells. `pcDefinePCell` creates a Pcell

Virtuoso Parameterized Cell Reference

Creating SKILL Parameterized Cells

master. This is the same result you get when you compile a graphical Pcell using *Pcell – Compile – To Pcell* in a layout window.

Each call to `pcDefinePCell` creates one Pcell master cellview. You can create one source code file for each Pcell or define several Pcells in one file. Either way, use the `load(filename)` command to load and run `pcDefinePCell`.

When you compile Pcell SKILL code, the compiler attaches the compiled code to the master cell. You do not need the source file containing the `pcDefinePCell` calls to be able to use your Pcell. But save the source file so you can revise it later.

The `pcDefinePCell` function passes three arguments to the Pcell compiler. Each argument is a list. For complete syntax and examples, see `pcDefinePCell` in the *Virtuoso Parameterized Cell SKILL Reference*.

Using the pcCellView Variable

`pcCellView` is an internal variable automatically created by `pcDefinePCell`. `pcCellView` contains the `dbID` (database identification) of the cell you are creating. Within the body of your Pcell code, use the `pcCellView` variable as the cellview identifier for which you create objects. For convenience, you might assign `pcCellView` to a variable with a shorter name; for example, `cv=pcCellView`.

Using the SKILL Operator ~> with Pcells

When you use the SKILL operator `~>` to access information about the master cell of a Pcell instance, you must look two levels above the instance. If you look at only one level above, you access information about the Pcell submaster.

For example, for the instance `I1` of the Pcell master `mux2`, you retrieve the database identity (`dbID`) of the `mux2` Pcell master with the following statement:

```
I1~>master~>superMaster
```

However, if you use the following statement:

```
I1~>master
```

then, you can retrieve the `dbID` for the Pcell submaster.

Defining Parameter Values

You must specify all parameters that affect the Pcell in the `pcDefinePCell` statement. You must also assign a default value for every parameter.

The values of parameters on a Pcell instance can be assigned in any of the following ways:

- On the instance itself

You can assign values to parameters when you place an instance. These values override any default values that might be defined. If you assign a value when you place an instance, the value is stored as a property on the instance and does not apply to other instances.

- Using CDF

You can create a Component Description Format (CDF) description to specify the value for a Pcell parameter. You can create CDFs for a cell or for a whole library. CDFs defined for a cell apply to all cellviews of that cell; for example, parameters shared by schematic and layout cellviews. CDFs defined for a library apply to all cells in the library.

Note: You must set the `?storeDefault` argument of the `cdfCreateParam` function to `yes` to store the default value of a user CDF parameter as a parameter attribute on the Pcell.

For example,

```
cdf = cdfCreateUserCellCDF( ddGetObj( samplelib pmos5v_mac ) )
cdfCreateParam(cdf
    ?name lpp
    ?defValue wire drawing
    ?type string
    ?display t
    ?prompt lpp
    ?storeDefault yes)
```

Here, the Pcell stores the default value of the CDF parameter `lpp`, that is, `wire drawing`.

- In the `pcDefinePCell` code

You must assign default values for parameters in the parameter declaration section of your `pcDefinePCell` statement. Values defined in the `pcDefinePCell` statement are stored with the Pcell master cellview.

In the following example, the parameters `width`, `length`, and `numGates` are defined and assigned defaults:

```
pcDefinePCell(
    list( ddGetObj( "pcellLib" ) "muxPcell" "layout" )
    ; parameters and their optional default values:
```

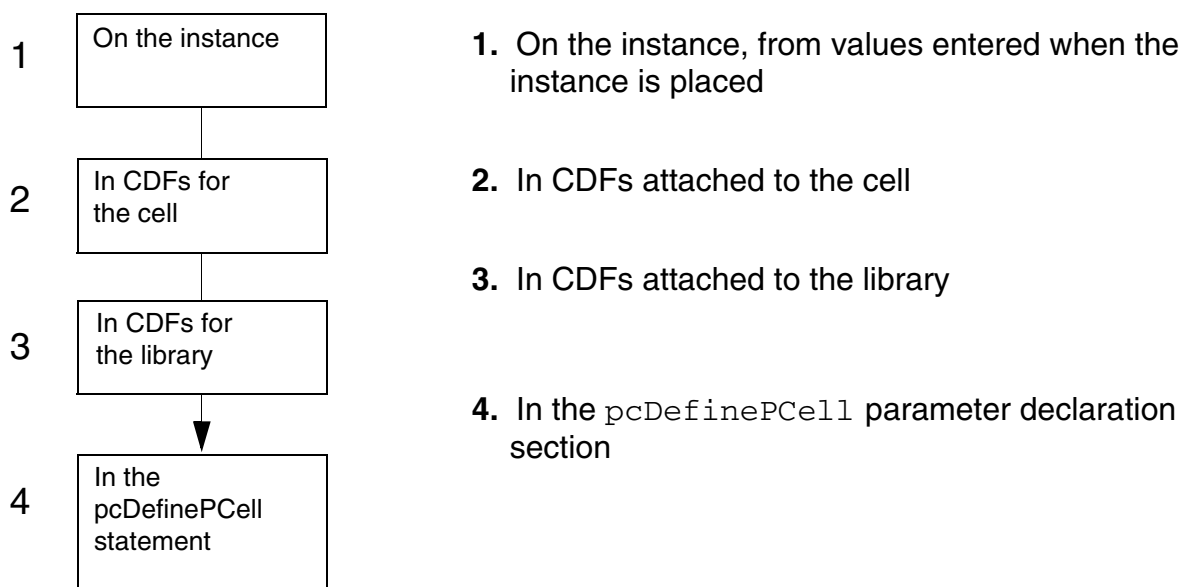
Virtuoso Parameterized Cell Reference

Creating SKILL Parameterized Cells

```
list(  
  ( width      "float"    1.0)  
  ( length     "float"    0.75)  
  ( numGates   "int"      1)  
) ;end of parameter list
```

The system looks for the values of parameters in a specific order. When the system finds a value, it stops looking, so where you define or assign values determine which value has highest priority.

The system looks for the value of parameters in the following order:



Important

Be careful when you choose where you set the default values. When the system finds a value for a parameter, it stops looking. Therefore, if you are setting a parameter value for a Pcell in the `pcDefinePCell` code (number 4 in the image) and also in the CDF of the Pcell (number 2 in the image), ensure that the settings match. Otherwise, the Create Instance and Edit Instance forms will not work properly. In such cases, you can use the *Debug CDF* tool to identify and fix the issue.

If you prefer to get parameter default values from a library or technology file, you can access properties within the parameter declaration section of the `pcDefinePCell` statement as shown in the following example.

Virtuoso Parameterized Cell Reference

Creating SKILL Parameterized Cells

Example of Setting Parameter Values

In this example, the first line accesses a library and the second line accesses a technology file.

```
; parameters and their default values:
list(
    (param1      ddGetObj("libName") ~> param1Default)
    (param2      techGetParam(techGetTechFile(pcCellView, param2Default))
) ;end of parameter list
```

Building Geometries

You can use the following procedures to build shapes in a Pcell. Refer to [*Virtuoso Design Environment SKILL Reference*](#) for details about these procedures.

```
dbCreateLine( d_cellView (tx_layer [t_purpose]) l_points)
dbCreatePath( d_cellView (tx_layer [t_purpose]) l_points x_widths
[t_pathStyle])
dbCreatePolygon( d_cellView (tx_layer [t_purpose]) l_points)
dbCreateRect( d_cellView (tx_layer [t_purpose]) l_bBox)
dbCreateEllipse( d_cellView (tx_layer [t_purpose]) l_bBox)
dbCreateDonut( d_cellView (tx_layer [t_purpose]) l_point x_outR x_holeR)
dbCreateLabel( d_cellView (tx_layer [t_purpose]) l_point t_label t_just
t_orient t_font x_height)
```

Follow the guidelines below when using dbCreate functions to build objects:

- *tx_layer* must specify either the mask layer number or a layer name. For example, to build a rectangle on the layer metal1, you might specify

```
dbCreateRect( pcCellView "metal1" list( x1:y1 x2:y2))
```
- The points list for paths and polygons must be a list of coordinate pairs, where each pair is a list containing two elements. The notation *x:y* is equivalent to `list(x y)`.
- Rectangles and ellipses are defined with bounding boxes. A bounding box is specified as a list of coordinate pairs, where the first pair refers to the lower left corner and the second pair refers to the upper right corner.
- You can use analog design-style labels, such as `cdsName()`, `cdsParam(1)`, `cdsParam(2)` to display information about the instance. Labels are controlled by the CDF label set attributes. Refer to [*Component Description Format User Guide*](#) for more information.

For the labels to be evaluated correctly, you must set `labelType` to "ILLabel" as shown in the following example:

Virtuoso Parameterized Cell Reference

Creating SKILL Parameterized Cells

```
label = dbCreateLabel(  
    pcCellView  
    '("text" "drawing")  
    x:y  
    "cdsName()" "  
    "lowerLeft" "R0" "stick" 5.0  
)  
label~>labelType = "ILLabel"
```

- The `pcFix` function lets you correct machine-dependent round-off error. If the argument of `pcFix` is no more than 0.001 away from an integer value, `pcFix` returns the integer value. Here are some examples:

```
pcFix(1.999999)    returns 2.0  
pcFix(2.000001)    returns 2.0  
pcFix(2.1)         returns 2.1
```

Creating Instances within Pcells

It is often useful to create instances of other cells within a Pcell. You can also create Pcell instances within a Pcell. For example, inside your Pcell code, you can create the following:

1. Get the cellview ID of the cell to add as an instance:

```
cellId = dbOpenCellViewByIdType( gt_lib t_cellName lt_viewName)
```

2. Create an instance of this cell:

```
inst = dbCreateInst( d_cellView d_master nil l_point t_orient 1)
```

3. If the instance is a Pcell, use

```
inst = dbCreateParamInst(  
    d_cellView d_master nil l_point t_orient 1  
    list(  
        list( param1Name "float" 10.)  
        list( param2Name "string" "some string" )  
    ) ; close list  
) ; close dbCreateParamInst
```

Building Nets, Terminals, and Pins

If you plan to use another tool that deals with layout connectivity, such as a router, device level editor, microwave extractor, or online extractor, you need to define the pins on your component layout cellviews. The pins in the layout cellview must match the pins in the schematic symbol cellview. The following steps show how to define pins in a layout Pcell.

Virtuoso Parameterized Cell Reference

Creating SKILL Parameterized Cells

1. Create the shape that will serve as the pin.

The shape is usually a rectangle.

Note: The shape cannot be a polygon.

```
fig = dbCreateRect( d_cellView tx_layer list( x1:y1 x2:y2))
```

2. Create the net to which the pin attaches.

In this example, the pin name `n1` matches the name of the corresponding pin in the schematic symbol for this cell:

```
net = dbCreateNet( d_cellView "n1")
```

3. Create a terminal on the same net.

The terminal must have the same name as the net and match the terminal type. In this example, the terminal type is `inputOutput`, the same terminal type as the corresponding pin in the schematic symbol:

```
trm = dbCreateTerm( d_net "n1" "inputOutput")
```

4. Create a pin:

```
pin = dbCreatePin( d_net d_fig "n1")
```

The pin database object connects the pin figure with the net. The pin name can match the terminal name but does not have to. In the example, the pin name `n1` matches the terminal name.

Within the Pcell, you can have multiple shapes that all belong to the same electrical terminal. Each shape would have a pin to associate it to the same net. In such cases, each pin is created on the same net and must have a unique pin name.

5. If your tool requires pins to have an access direction, define the access direction:

```
pin~>accessDir = ' ( "top" "left")
```

The access direction is a list identifying the correct sides of the pin shape for connection.

Accessing Technology File Data

You can access nominal and minimum dimensions and other technology file information from within a Pcell, using `tech` procedures. Use `tech` procedures to keep process-specific information, such as layer-to-layer spacings, localized to the technology file. Make sure rules you want to read from the technology file are defined there before you load your Pcell code; otherwise, the Pcell compilation fails.

The following three examples show how to get data from the technology file.

Example 1

Access a single-layer spacing rule, such as the minimum width:

```
tf = techGetTechFile(d_cellView)
d = techGetSpacingRule(tf "minWidth" "layerName")
```

Example 2

Access a two-layer property, such as the minimum spacing between two layers:

```
d = techGetSpacingRule(tf "minSpacing" "layer1" "layer2")
```

Example 3

Access your own user-created property, such as the layer name for shapes:

```
techGetLayerProp(tf "layerName" "propName")
```

Variable Scoping

The code you include within the `pcDefinePCell` function can access only local variables and the input parameters to the Pcell. Procedures called from Pcells, however, can access global variables. To prevent unintended access of global variables, use a `let` or `prog` construct in your procedures to define local variables.

In some situations, you might want to access global variables. If so, see the example about accessing global variables in [Debugging SKILL Pcells](#) on page 179.

There is an example illustrating how to include user-defined procedures in a Pcell in [Example 3: Calling User-Defined Procedures](#) on page 175.

Schematic Symbol Pcells

The Virtuoso Schematic Editor design entry tool is built using the same graphic editor environment as the other Virtuoso tools. You can build schematic symbols as Pcells using SKILL. This makes it possible to have schematic symbols that change shape, depending on the parameters of the specific symbol instance.

The following example shows how to create a schematic symbol Pcell with the two parameters, `inputLeads` and `outputLeads`.

Virtuoso Parameterized Cell Reference

Creating SKILL Parameterized Cells

```
pcDefinePCell(  
  list( ddGetObj( "pcellLib") "MyPcell" "symbol" "schematicSymbol")  
  ; formal parameters  
  list(  
    (inputLeads      "string" "1")  
    (outputLeads     "string" "2")  
  ) ;end of parameter list  
)
```

In the above example, a schematic Pcell called `MyPcell` is created with two output leads and one input lead. You can change the size of the symbol by changing the values of the parameters.

For more information about debugging a schematic Pcell, refer to [*Pcell IDE - Quick Start Guide*](#).

Advantages of ROD Functions in Pcell

Major advantages of using ROD to create Pcells are:

- A single ROD function call can complete a task that otherwise would require several lower-level SKILL function calls. For example, creating a pin required a series of low-level SKILL function calls, but with ROD, you can use a single function to create a shape and designate it as a pin. You can also create a derived object, that is, create an object from an existing object just by specifying the size of the new object.

In an object created using a ROD function, multiple handles for the same point are saved in memory in comparison to a similar object created using a database function. It saves a great deal of time and effort in calculating and tracking coordinates of new objects and aligning the objects.

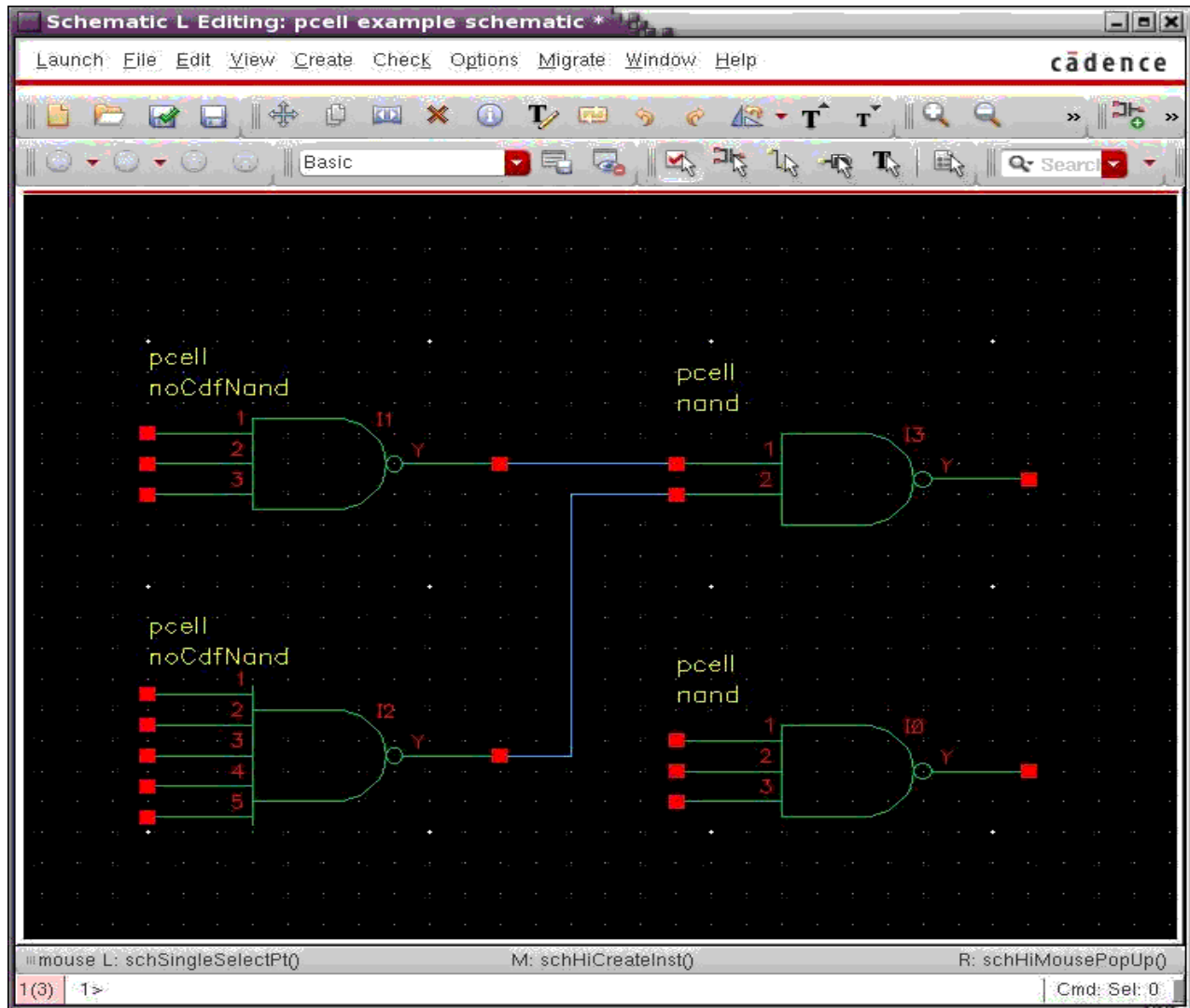
- You can base the Pcell parameters on rules from your technology file, making the Pcells tolerant of changes to your technology.
- You can assign point handles to Pcell parameters that let you update the parameters interactively by stretching the Pcell in Virtuoso Layout Editor.

Virtuoso Parameterized Cell Reference

Creating SKILL Parameterized Cells

Using a Variable Number of Terminals

With Pcell symbols, it is possible to specify a variable number of terminals. The next figure shows the instances of variable inputs for the two nand Pcells.



The only limitation is that the netlisting software you use for simulation must be able to handle a variable number of terminals on these cells. For example, the microwave environment can handle a variable number of terminals on these cells.

If the application you want to use can handle a variable number of terminals, you can implement digital gates using Pcells. For example, instead of creating four or more NOR gate cells, each with a different number of inputs, you could create a single NOR gate cell with a cyclic parameter field that specifies the number of inputs.

Virtuoso Parameterized Cell Reference

Creating SKILL Parameterized Cells

Virtuoso Parameterized Cell Reference

Creating SKILL Parameterized Cells

Creating SKILL++ Parameterized Cells

The Pcell source is typically written in SKILL, which is a multi-paradigm language. However, you can also write the Pcell source in SKILL++ along with the existing APIs.

Advantages of SKILL++ Pcells

You can create individual Pcells using SKILL++ classes and methods that leverage the Object Oriented Programming (OOP) concepts. Using multiple class inheritance, it is possible to create a new Pcell by inheriting from other Pcell classes.

Using SKILL++ Pcells renders the following advantages of OOP:

- A clear modular structure for programs – You can define abstract data types with hidden implementation details and using a well-defined interface of the unit.
- Quick maintenance and modification of the existing code – You can reuse the existing code to create new objects.
- A good framework for code libraries – The supplied software components can be quickly adapted and modified by the programmer.

Recommended and Supported SKILL++ Functions for Pcells

The following pc functions can be used for SKILL++ Pcells.

- pcSetParamSlotsFromMaster
- pcDefineParamSlot
- pcGetParamSlotType
- pcGetParamSlotValue
- pcSetParamSlotValue

- `pclsParamSlot`
- `pcGetDefaultParamsFromClass`

Steps to Create SKILL++ Pcell

The following steps illustrate creation of a simple SKILL++ Pcell using the existing `pcDefinePCell` API, which has the Pcell source body implemented in SKILL++. To view the complete sample code, see the [SKILL++ Pcell Samples](#) section.

1. Define a base class (`PcellParam`) with a single slot for holding the Pcell master ID, which is the same as the `pcCellView` variable used in SKILL Pcell.
2. Define a class to represent the Pcell. This class is defined by inheriting from `PcellParam` (or using `PcellParam` as its superclass).
3. Define a method (`setPcellParams`) to transfer the Pcell parameter values to the slots of the Pcell class. This ensures that the Pcell code written in SKILL++ has full access to the same set of Pcell parameters.
4. Define a method (`draw`) to construct the Pcell.
5. Implement the code to call `pcDefinePCell` to create the supermaster.

SKILL++ Pcell Samples

You need to perform the following steps to setup SKILL++ Pcells:

1. Load `generic.ils`, `PcellParam.ils`, `CORE.ils`, `RING.ils`, and `WRAP.ils`. These files are required for successful Pcell evaluation.
2. Load `genPcell.il`. This file is only needed to create the Pcell supermasters.
3. Load `genPcDef.il` only if you need to create the `pcDefinePCell` code from Pcell class.

Refer to the following sample code to setup SKILL++ Pcell.

```
; Start of file - generic.ils
; Defines the generic function for draw.
; This is placed in its own file as this should be loaded
; once as redefining a generic function will remove all methods
; defined by previous defgeneric form
(defgeneric draw (device)
  t
)
; End of file - generic.ils

; Start of file - PcellParam.ils
; Base class for all Pcells.
; cvId will hold the pcCellView value
defclass( PcellParam ()
  (
    (cvId @initarg cvId)
  )
)
; A simple method to populate the Pcell parameters from the supermaster to
; a Pcell device's slots
defmethod( setPcellParams ((device PcellParam) cv)
  when( cv && dbIsId(cv)
    setSlotValue(device 'cvId cv)
    foreach( param cv~>parameters~>value
      setParamValue(device concat(param~>name) param~>value)
    )
  )
)
; A function to define a Pcell parameter stored as a Pcell class' slot.
; The optional argument _isParam is set to 't to indicate this parameter is
; a Pcell parameter
defun( defineParam (g_type g_value @optional (_isParam t))
  list(nil 'type g_type 'value g_value 'isParam _isParam)
)
; A method to get Pcell parameter's type
defmethod( getParamType ((device PcellParam) (propName symbol))
  slotValue(device propName)->type
)
; A method to get Pcell parameter's value
defmethod( getParamValue ((device PcellParam) (propName symbol))
  slotValue(device propName)->value
)
; A method to set Pcell parameter's value
```

Virtuoso Parameterized Cell Reference

Creating SKILL++ Parameterized Cells

```

defmethod( setParamValue ((device PcellParam) (propName symbol) val)
  slotValue(device propName)->value = val
)
; A method to check is the given name a Pcell parameter or not. This is
; based on the setting of the isParam attribute
defmethod( isParam ((device PcellParam) (propName symbol))
  slotValue(device propName)->isParam
)
; A method to get a list of Pcell parameters with their names, types and
; values
defmethod( getPcellParams ((device PcellParam))
  let((params)
    params = setof( p device->? isParam(device p ))
    params = foreach( mapcar p params
      list( p
        getParamType(device p)
        getParamValue(device p)
      )
    )
  )
)
; End of file - PcellParam.ils

; Start of file - CORE.ils
; Defines a CORE class inheriting from PcellParam
; Note, parameter created via defineParam will be treated
; as a Pcell parameter; otherwise it is treated as a normal
; class slot (e.g. coreBBox is not a Pcell parameter)
; You can replace the library name mentioned here with your own library.
defclass( CORE (PcellParam)
  (
    (cyanW @initform defineParam("float" 0.6))
    (cyanL @initform defineParam("float" 0.2))
    (greenW @initform defineParam("float" 0.2))
    (greenL @initform defineParam("float" 0.8))
    (coreBBox @initarg coreBBox)
  )
)
; Draw a simple cross represented by two rectangles
defmethod( draw ((device CORE))
  let((cv cyanW cyanL greenW greenL rectId llx lly urx ury)
    cyanW = getParamValue(device 'cyanW)
    cyanL = getParamValue(device 'cyanL)
    greenW = getParamValue(device 'greenW)
    greenL = getParamValue(device 'greenL)
    ; layers are choosen for their color's visibility
    ; You can replace the layer names mentioned here with the layer names
    ; present in your library.
    cv = slotValue(device 'cvId)
    rectId = dbCreateRect(cv
      list("Poly" "drawing") list(0:0 greenL:greenW))
    llx = 0.5 * greenL - 0.5 * cyanL
    lly = 0.5 * greenW - 0.5 * cyanW
    urx = 0.5 * greenL + 0.5 * cyanL
    ury = 0.5 * greenW + 0.5 * cyanW
    rectId = dbCreateRect(cv
      list("Vial" "drawing") list(llx:lly urx:ury))
    setSlotValue(device 'coreBBox list( 0.0:lly greenL:ury))
    callNextMethod()
  )
)

```

Virtuoso Parameterized Cell Reference

Creating SKILL++ Parameterized Cells

```
; Returns CORE's bounding box which is stored in the coreBBox slot
defmethod( getCoreBBox ((device CORE))
  slotValue(device 'coreBBox)
)
; End of file - CORE.ils

; Start of file - RING.ils
; Defines a RING class inheriting from PcellParam
defclass( RING (PcellParam)
  (
    (ringW @initform  defineParam("float" 0.1))
    (ringS @initform  defineParam("float" 0.1))
  )
)
; Draw a polygon that wraps around the device's coreBBox with a given
; spacing value and width of the polygon
defmethod( draw ((device RING))
  let((cv ringS ringW coreBBox llx lly urx ury pts ring)
    ringS = getParamValue(device 'ringS)
    ringW = getParamValue(device 'ringW)
    coreBBox = getCoreBBox(device)
    llx = xCoord( lowerLeft(coreBBox))
    lly = yCoord( lowerLeft(coreBBox))
    urx = xCoord( upperRight(coreBBox))
    ury = yCoord( upperRight(coreBBox))
    pts = list(
      llx-ringS:lly-ringS ; points on inner edges
      urx+ringS:lly-ringS
      urx+ringS:ury+ringS
      llx-ringS:ury+ringS
      llx-ringS:lly-ringS-ringW ; extending to outer edge
      llx-ringS-ringW:lly-ringS-ringW; points on outer edges
      llx-ringS-ringW:ury+ringS+ringW
      urx+ringS+ringW:ury+ringS+ringW
      urx+ringS+ringW:lly-ringS-ringW
      llx-ringS:lly-ringS-ringW)
    ; layer is chosen for its color's visibility
    ; You can replace the layer names mentioned here with the layer names
    ; present in your library.
    cv = slotValue(device 'cvId)
    ring = dbCreatePolygon( cv list("Via2" "drawing") pts)
    callNextMethod()
  )
)
; This method is called when RING is used on its own.
; Just return a empty box
defmethod(getCoreBBox ((device RING))
  list(0:0 0:0)
)
; End of file - RING.ils

; Start of file - WRAP.ils
; Defines a wrapped core class that inheriting both CORE and RING classes
defclass( WRAP (CORE RING)
  ()
)
; The draw method for WRAP is just a simple call to callNextMethod and due
; to the specificity of WRAP's superclasses, the draw methods for CORE and RING
; will be called in this order
defmethod(draw ((device WRAP) )
  callNextMethod()
)
```

Virtuoso Parameterized Cell Reference

Creating SKILL++ Parameterized Cells

```
; End of file - WRAP.ils
; Start of file - genPcell.il
; Code to create a Pcell supermaster for CORE
pcDefinePCell(
  list(ddGetObj("PDK_devices") "CORE" "layout")
  (
    ( cyanW "float" 0.6 )
    ( greenL "float" 0.8 )
    ( greenW "float" 0.2 )
    ( cyanL "float" 0.2 )
  )
  let((pcell)
    pcell = makeInstance('CORE)
    setPcellParams(pcell pcCellView)
    draw(pcell)
  )
)
; Code to create a Pcell supermaster for RING.
pcDefinePCell(
  list(ddGetObj("PDK_devices") "RING" "layout")
  (
    ( ringW "float" 0.1 )
    ( ringS "float" 0.1 )
  )
  let((pcell)
    pcell = makeInstance('RING)
    setPcellParams(pcell pcCellView)
    draw(pcell)
  )
)
; Code to create a Pcell supermaster
pcDefinePCell(
  list(ddGetObj("PDK_devices") "WRAP" "layout")
  (
    ( ringW "float" 0.1 )
    ( greenL "float" 0.8 )
    ( greenW "float" 0.2 )
    ( cyanL "float" 0.2 )
    ( cyanW "float" 0.6 )
    ( ringS "float" 0.1 )
  )
  let((pcell)
    pcell = makeInstance('WRAP)
    setPcellParams(pcell pcCellView)
    draw(pcell)
  )
)
; End of file - genPcell.il

; Start of file - genPcDef.il
; Based on specified class name, lib name, cell name, view name, and draw
; function name to generate Pcell definition file in user specified directory.
; genPcellCodeFromClass(
;   s_className t_libName t_cellName t_viewName t_pcFilename t_drawFuncName)
; For example,
; genPcellCodeFromClass(
;   'WRAP "PDK_devices" "WRAP" "layout" "./pcWrap.il" "draw")
procedure( genPcellCodeFromClass(
  className libN cellN viewN filePathName drawFuncName)
  let((oport params device str dirpath)
    when( !filePathName || filePathName == ""
```

Virtuoso Parameterized Cell Reference

Creating SKILL++ Parameterized Cells

```
warn("File path name is empty.\n")
return()
)
str = rindex(filePathName "/" )
when( str
    dirpath = substring(filePathName 1 strlen(filePathName) - strlen(str))
    when( !isDir(dirpath)
        warn("Directory( %s ) does not exists.\n")
        return()
    )
)
)
oport = outfile(filePathName "w")
fprintf(oport "pcDefinePCell(\n\ttlist(ddGetObj(\"%s\") \"%s\" \"%s\")\n"
    libN cellN viewN)
; Output parameters section
device = makeInstance(concat(_className))
fprintf(oport "\t(\n")
params = getPcellParams(device)
foreach( param params
    case( cadr(param)
        ("int"
            fprintf(oport "\t\t( %s \"int\" %d )\n" car(param) caddr(param))
        )
        ("float"
            fprintf(oport "\t\t( %s \"float\" %f )\n" car(param) caddr(param))
        )
        ("string"
            fprintf(oport "\t\t( %s \"string\" \"%s\" )\n" car(param) caddr(param))
        )
        ("boolean"
            fprintf(oport "\t\t( %s \"string\" %L )\n" car(param) caddr(param))
        )
        (("ILLlist" "ilList")
            fprintf(oport "\t\t( %s \"ILLlist\" %L )\n" car(param) caddr(param))
        )
        (t
            warn("Unsupported type %s on parameter %s skipped\n"
                cadr(param) car(param))
        )
    )
)
)
fprintf(oport "\t)\n")
; Output the standard Pcell code section
fprintf(oport "\t\tlet((pcell)\n")
fprintf(oport "\t\t\ttcell = makeInstance('%s')\n" _className)
fprintf(oport "\t\t\ttsetPcellParams(pcell pcCellView)\n")
fprintf(oport "\t\t\t%s(pcell)\n" drawFuncName)
fprintf(oport "\t)\n")
fprintf(oport ")\n")
close(oport)
; Auto generate Pcell superMaster by user defined Pcell class
; load(filePathName)
)
)
; End of file - genPcDef.il
```

Virtuoso Parameterized Cell Reference

Creating SKILL++ Parameterized Cells

Creating Graphical Parameterized Cells

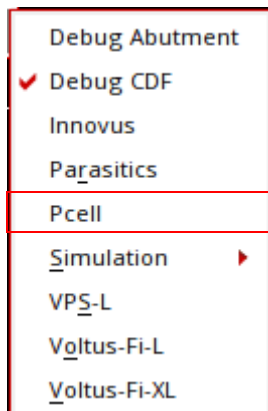
This chapter describes the Virtuoso[®] parameterized cell software, which provides a graphical user interface that lets you create parameterized cells (Pcells) for placement in design layouts. You define the master cell and its parameters with this tool and can change parameter values for each instance you create in a layout cellview.

This chapter contains the following information:

- [Adding the Pcell Menu](#)
- [Stretch Commands](#)
- [Conditional Inclusion Commands](#)
- [Repetition Commands](#)
- [Parameterized Shapes Commands](#)
- [Repetition Along Shape Commands](#)
- [Reference Point Commands](#)
- [Inherited Parameters Commands](#)
- [Parameterized Layer Commands](#)
- [Parameterized Label Commands](#)
- [Parameterized Property Commands](#)
- [Parameters Commands](#)
- [Compile Commands](#)
- [Make Ultra Pcell Command](#)
- [Customizing the Pcell Compiler](#)
- [Creating Complex Pcells](#)
- [Converting Graphical Pcells to SKILL Code](#)

Adding the Pcell Menu

To add the *Pcell* menu, select *Launch – Plugins – Pcell* from any VLS, VSE, or VSE Symbol editor window.



Stretch Commands

Stretch parameters change the size of all objects in a cellview that are not excluded from the stretch. You can stretch objects horizontally, vertically, or both.

- [Drawing Stretch Control Lines](#)
- [Defining Stretch Parameters](#)
- [Specifying Stretch Directions](#)
- [Specifying a Reference Dimension](#)
- [Setting Minimum and Maximum Values](#)
- [Stretching Paths](#)
- [Using Stretch with Repetition](#)
- [Using Stretch with Conditional Inclusion](#)
- Using the *Stretch* menu, including
 - ❑ Stretching objects horizontally (see [“Stretch in X”](#) on page 65)
 - ❑ Stretching objects vertically (see [“Stretch in Y”](#) on page 66)

Virtuoso Parameterized Cell Reference

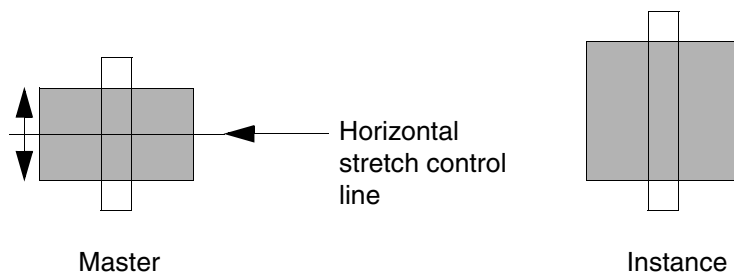
Creating Graphical Parameterized Cells

- ❑ Stretching a selected set of objects instead of all objects in the cellview (see [“Qualify”](#) on page 67)
- ❑ Changing parameters assigned to stretch control lines (see [“Modify”](#) on page 69)
- ❑ Changing the location of the stretch control lines and their parameters (see [“Redefine”](#) on page 70)

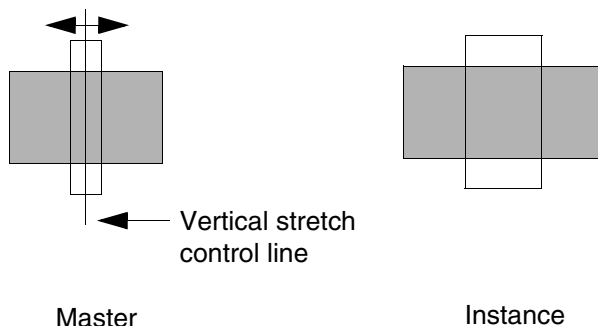
Drawing Stretch Control Lines

To add a stretch parameter to your cellview, you must first draw a stretch control line. Stretch control lines determine where to begin the stretch and which direction to stretch. Horizontal stretch control lines control vertical stretching, and vertical stretch control lines control horizontal stretching.

A horizontal stretch control line controls whether objects stretch upward from the line, downward from the line, or both.



A vertical stretch control line controls whether objects stretch to the right of the line, to the left of the line, or both.



You can use as many stretch control lines as you want in a single cellview. The Virtuoso® Parameterized Cell (Pcell) program treats each stretch control line independently.

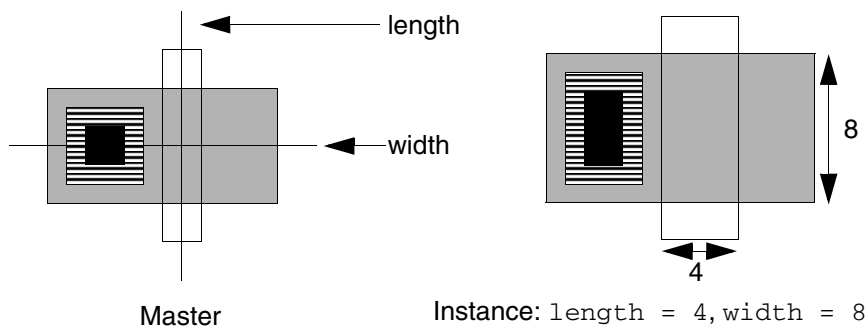
Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

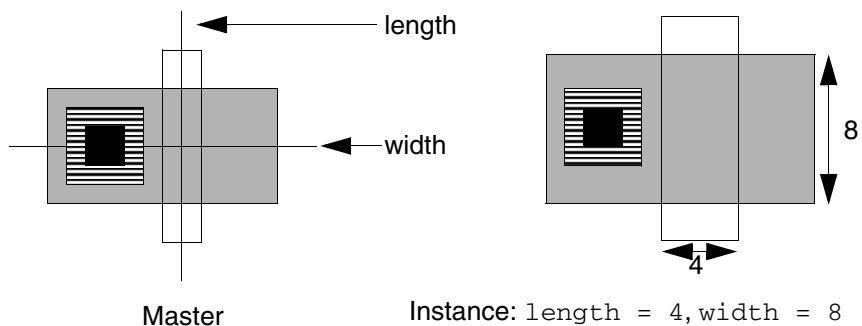
Important

Stretch control lines are always extrapolated automatically from their end points to the edges (right to left or left to right) of the cell.

The following example shows two stretch control lines that stretch a transistor horizontally and vertically. These stretch control lines are named `length` and `width`.



You can limit the effect of a stretch control line by using the *Qualify* command. In the following example, you can exclude the contact and its surrounding metal from the stretch so that only the poly and diffusion are stretched.



Stretch Control Line Rules

When you draw a stretch control line, follow these rules:

- Draw stretch control lines with orthogonal snap (only horizontal and vertical line segments).

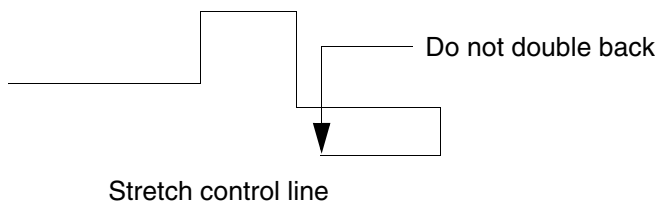
You can set the default snap mode using *Design – Options – Display*.

- Do not reverse the direction of a stretch control line.

Virtuoso Parameterized Cell Reference

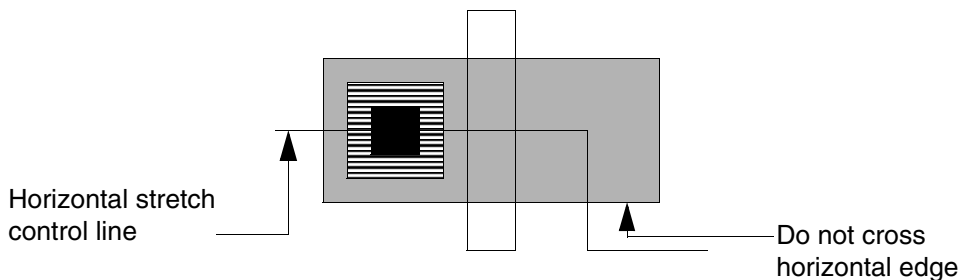
Creating Graphical Parameterized Cells

You can draw a stretch control line around objects you want to exclude from the stretch, but the line cannot double back on itself. Horizontal stretch control lines must be drawn from either right to left or left to right. Vertical stretch control lines must be drawn from either top to bottom or bottom to top. Once you start drawing a stretch control line in one direction, you cannot go back.



- Although horizontal stretch control lines can contain vertical segments, do not draw horizontal stretch control lines that cross horizontal edges of objects included in the stretch.

Vertical stretch control lines cannot intersect vertical edges of objects included in the stretch. The first segment that you draw determines if the stretch control line is a horizontal or vertical stretch control line.



After drawing a stretch control line, you must define it. You can then refer to the stretch control line in other parameter definitions.

Deleting a Stretch Control Line

After you have made the stretch layer valid, you can delete stretch control lines.

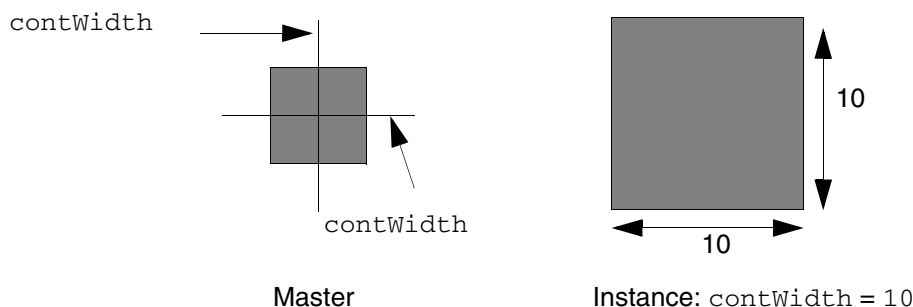
To delete the stretch control line,

1. Select the stretch control line.
2. In the layout editor, choose *Edit – Delete*.

Defining Stretch Parameters

After you draw a stretch control line, you must give it a definition. If the definition is a name, such as `width`, the system prompts you to enter a value for `width` when you place an instance of the Pcell. The definition can be a SKILL expression that references other parameter definitions in the Pcell, such as `gates*2`. The system uses the value for the repetition parameter `gates` to compute the value for the stretch parameter.

If you want to apply the same parameter value to more than one stretch line, use the same definition. For example, when you use the definition `contWidth` for both horizontal and vertical stretch control lines, the Pcell program stretches the following square contact the same amount in each direction so the contact remains square.



Specifying Stretch Directions

When you define a stretch control line, you must specify the stretch direction. Choose *left*, *right*, or *left and right* for a horizontal stretch control line. Choose *up*, *down*, or *up and down* for a vertical stretch control line. If an object is on one side of the stretch control line, it moves rather than stretches. Every vertex of the object stretches the same amount, which has the effect of moving the object. Objects move only if they lie entirely on the side of the stretch direction.

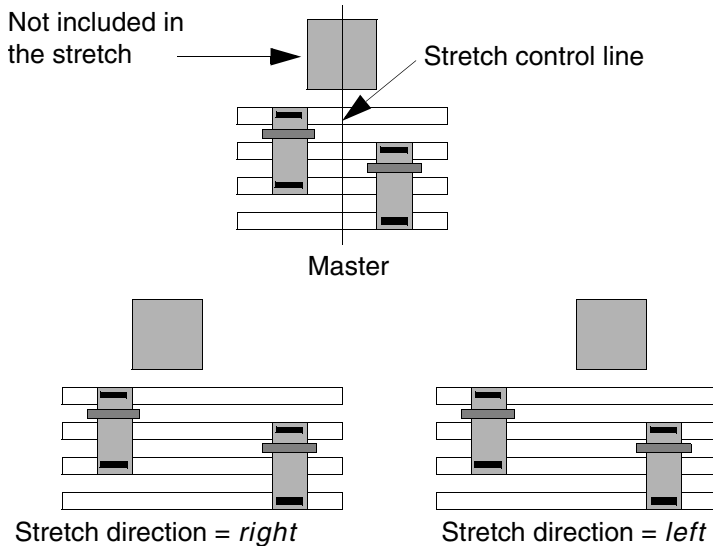
Stretching in One Direction

If the stretch direction is *right* or *left*, the Pcell program repositions every vertex to the right or left of the stretch control line by the amount specified. The stretch affects all objects

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

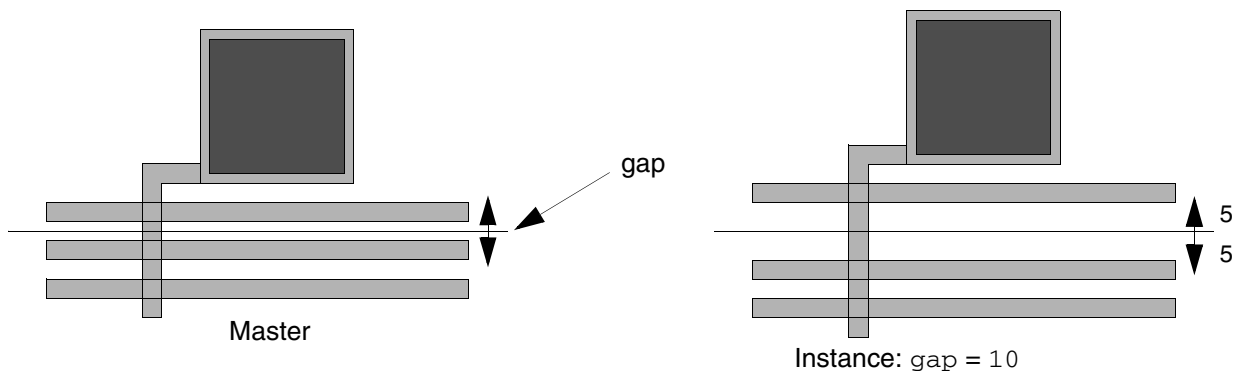
(rectangles, paths, polygons). Objects in repetition groups are excluded from the stretch by default.



Stretching in Two Directions

If the direction is *left and right* or *up and down*, the objects stretch or move by half the parameter value on each side of the stretch control line.

In the following example, the stretch control line is named `gap`, the direction of the stretch is *up and down*, and the parameter value is 10. Objects that lie entirely above the stretch control line move up half the parameter value. Objects that lie below the stretch control line move down half the parameter value. Objects crossed by the stretch control line stretch up half the parameter value and down half the parameter value.

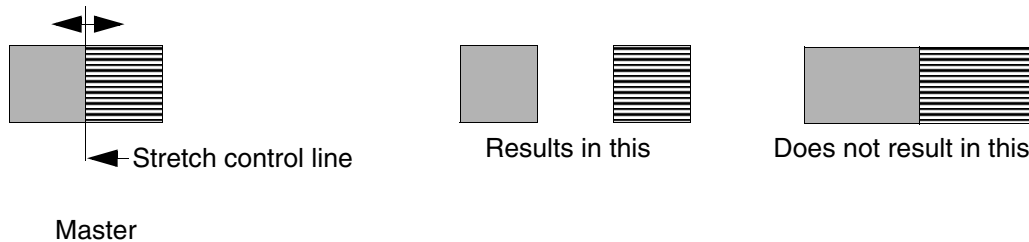


Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

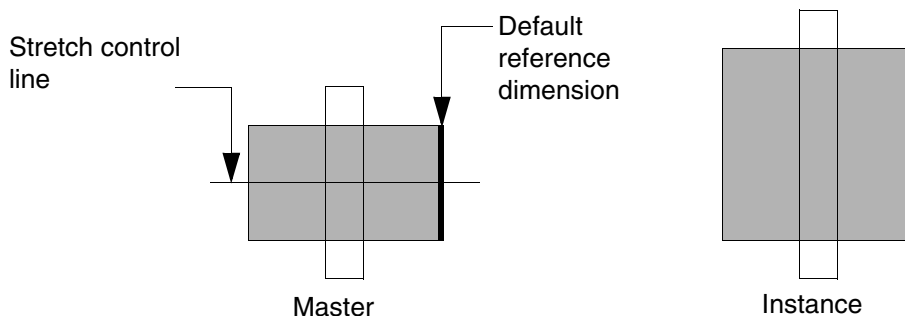
Stretching Edges Coincident with the Stretch Control Line

Objects are stretched only if they are crossed by a stretch control line. If the edge of an object is coincident with a stretch control line, the object moves.



Specifying a Reference Dimension

All stretch control lines use a *reference dimension* to determine the amount of stretch. By default, the reference dimension is the length of the shortest edge crossed by the stretch control line.



When you place an instance, you specify a parameter value. The parameter value is the width or length of the object after the stretch is complete, not the amount to stretch. The Pcell program computes the amount of stretch by subtracting the reference dimension from the parameter value. If the parameter value is greater than the reference dimension, the objects are enlarged. If the parameter value is less than the reference dimension, the objects are compressed.

You can specify a reference dimension or use the default reference dimension. To specify your own reference dimension, use a measurement from an important object that is included in the stretch, such as the length or width of a gate.

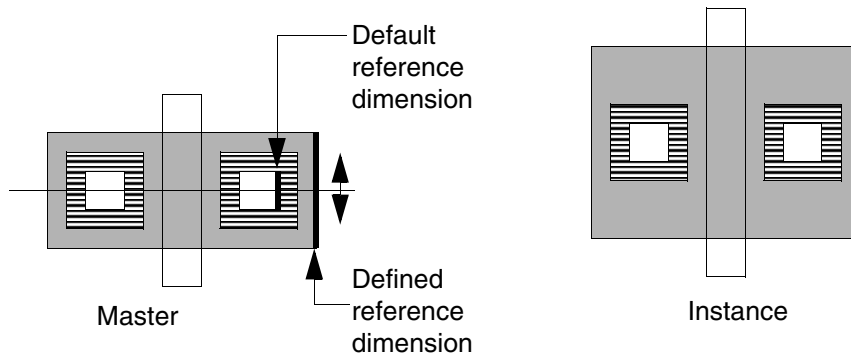
For example, if you want to expand the diffusion layer in a transistor but your stretch control line crosses the contacts, the default reference dimension is the length of the edge of the contact, the shortest edge crossed by the stretch control line. If you exclude the contacts from

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

the stretch, you do not want to use their dimension for the reference dimension. If the contacts are included in the stretch, but you want the parameter `width` to refer to the width of the gate, set the reference dimension to equal the length of the diffusion edge of the transistor.

If the contacts and their surrounding metal are excluded from the stretch and the direction of stretch is *up and down*, the contacts remain centered while the diffusion and poly stretch.



The system stretches all objects whose dimensions are equal to the reference dimension until their dimensions are equal to the value entered for the stretch parameter. Objects with other dimensions are stretched accordingly. You do not want to use the dimension of the contact for the reference dimension because the system would stretch the contact until its dimension equaled the value entered for `width`.

The system also uses the reference dimension value as a default value for stretch parameters. You control how the system does this with the `stretchPCellApplyToName` environment variable.

Setting Minimum and Maximum Values

You can set minimum and maximum values for a stretch parameter (or for all the parameters in a stretch expression) so that an object is not stretched more or less than the design rules allow. For example, you might define a transistor with a minimum channel length of 1.25 microns and a maximum length of 10 microns. If you try to place an instance of the cell with a channel length less than 1.25 or greater than 10, the Pcell program reverts to the previous value of the parameter before generating the instance.

If you do not set a minimum value, the system uses a default of 0.0 and does not perform minimum range checking. If you do not set a maximum value, the system uses a default of 0.0 and does not perform maximum range checking. If you set a maximum value that is less than the reference dimension, the system ignores the maximum value.

Applying Default, Minimum, and Maximum Values

You specify the *Reference Dimension*, *Minimum Value*, and *Maximum Value* fields in the Stretch in Y and Stretch in X forms.

For the *Name or Expression for Stretch* field, you specify either the name of a parameter (variable), such as `length`, or a SKILL expression. The SKILL expression references one or more parameters of the Pcell.

The `stretchPCellApplyToName` environment variable controls how the system applies the reference dimension (default) and minimum and maximum values. You can set the `stretchPCellApplyToName` environment variable in your `.cdsenv` file or in the Command Interpreter Window (CIW). The default is `t`.

Note: The `stretchPCellApplyToName` environment variable is available in releases starting with 4.4.6 and in the following releases:

4.4.5 100.7
4.4.3 100.76
4.4.2 100.18

Using `stretchPCellApplyToName` Equal to `t` (the default)

When `stretchPCellApplyToName` is set to `t`, the software behaves as described in this section.

■ Reference dimension

The system uses the number in the *Reference Dimension* field as the default value for the parameter name you specify in the *Name or Expression for Stretch* field. When you specify an expression, the system uses the reference dimension number as the default value for every parameter in the expression.

■ Minimum and maximum value

The range specified by values in the *Minimum Value* and *Maximum Value* fields applies to the parameter name you specify in the *Name or Expression for Stretch* field.

When you specify an expression, the system performs minimum-maximum range checking on every parameter in the expression and on the result obtained from evaluating the expression.

When you place a Pcell instance and change the default value for a parameter to a value that is outside of the specified minimum-maximum range, the system changes the out-of-range

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

parameter back to its previous value when you press the `Tab` key or click to place the instance. The system also issues a warning message in the CIW.

When you place a Pcell instance and change the default value for the parameters in an expression, and the value of the expression falls outside of the minimum-maximum range, the system uses the specified minimum or maximum value as the value of the expression. The system does not issue a warning message.

For example, the Stretch in X form defines a Pcell master as follows:

<i>Name or Expression for Stretch</i>	<code>parm_1 + (2.0 * parm_2) + 1.0</code>
<i>Reference Dimension</i>	<code>1.0</code>
<i>Minimum Value</i>	<code>1.0</code>
<i>Maximum Value</i>	<code>5.0</code>

When you select the Pcell to place an instance and change the value of `parm_2` to 6.0, as soon as you press the `Tab` key or click to place the instance, the system issues a warning in the CIW and changes the value of `parm_2` back to 1.0.

When you select the Pcell to place an instance and change the value of `parm_2` to 4.0, the system accepts the value until it evaluates the expression when you click to place the instance. Because the expression evaluates to 10.0, which is more than the maximum of 5.0, the system uses the maximum value of 5.0 instead of the result of the expression.

Using `stretchPCellApplyToName` Equal to `nil`

When `stretchPCellApplyToName` is set to `nil` and you specify a single parameter name in the *Name or Expression for Stretch* field, the software behaves the same as when `stretchPCellApplyToName` is set to `t` (as described previously).

When `stretchPCellApplyToName` is set to `nil` and you specify an expression in the *Name or Expression for Stretch* field, the software behaves similarly to when `stretchPCellApplyToName` is set to `t` (as described previously), with the following exceptions:

- Reference dimension

The system uses 0.0 as the default value for all parameters named in the *Name or Expression for Stretch* field.

- Minimum and maximum value

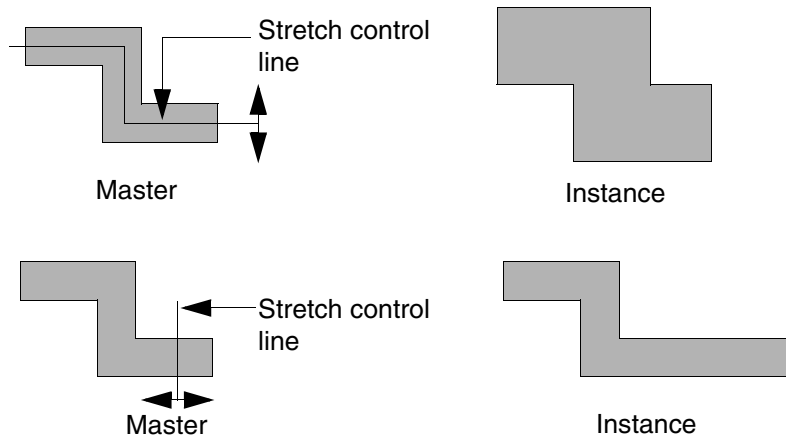
The system does not perform minimum-maximum range checking on the parameters in the expression but does perform minimum-maximum range checking on the result obtained from evaluating the expression.

Stretching Paths

You can stretch a path in two ways:

- Stretch path length using a stretch control line that bisects one or more of the path segments in a perpendicular direction.
- Stretch path width by setting the direction of the stretch to either *up and down* or *right and left* and by using a stretch control line that lies on every path vertex, including the startpoints and the endpoints.

The first figure illustrates a path whose width is controlled by the stretch control line if the direction is *up and down*. The second figure illustrates a path whose length is controlled by the stretch control line if the direction is *right and left*.



Using Stretch with Repetition

You can use stretch parameters with repetition parameters to

- Stretch repeated objects
- Repeat depending on the amount of stretch
- Stretch depending on the number of repetitions

Virtuoso Parameterized Cell Reference

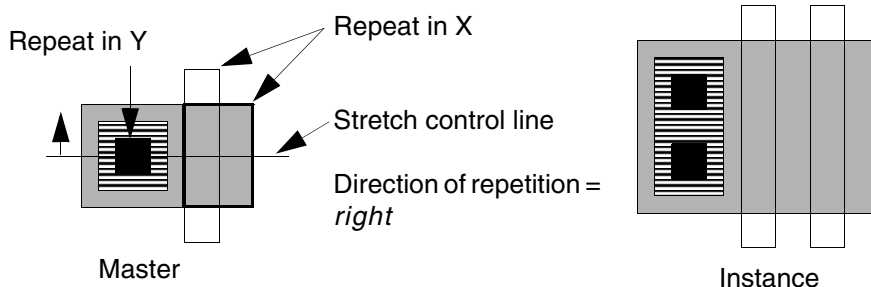
Creating Graphical Parameterized Cells

You can combine stretch parameters with repetition parameters in the same Pcell. By default, stretching takes place before repetition. To repeat objects before stretching them, you must specify the stretch control line as dependent on the repetition parameter.

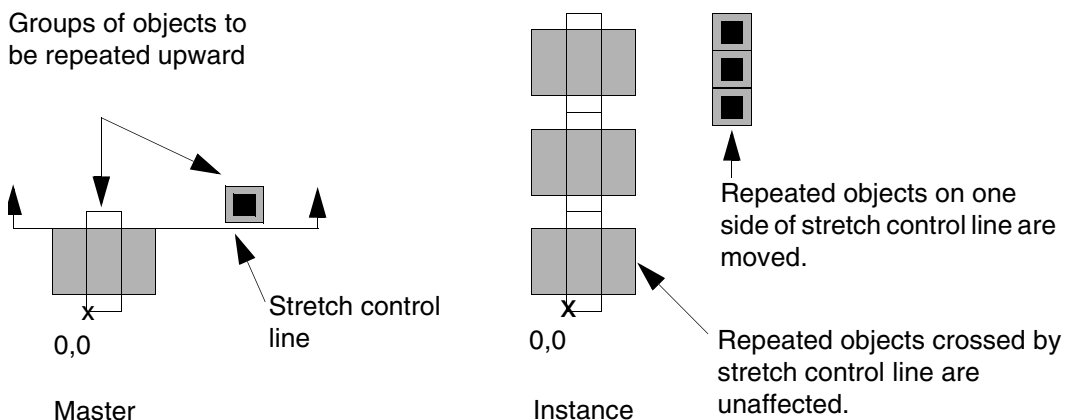
Using Stretch Parameters with Repeated Objects

When a stretch control line crosses an object to be repeated, the effect depends on the stretch direction and the direction of the repetition.

- If the stretch direction is perpendicular to the repetition direction, the object or group of objects is stretched first and then repeated.



- If the stretch direction is parallel to the direction of repetition, the object or group of objects are excluded from the stretch by default. The repetition group is not affected unless all objects in the group lie on the same side of the stretch control line.

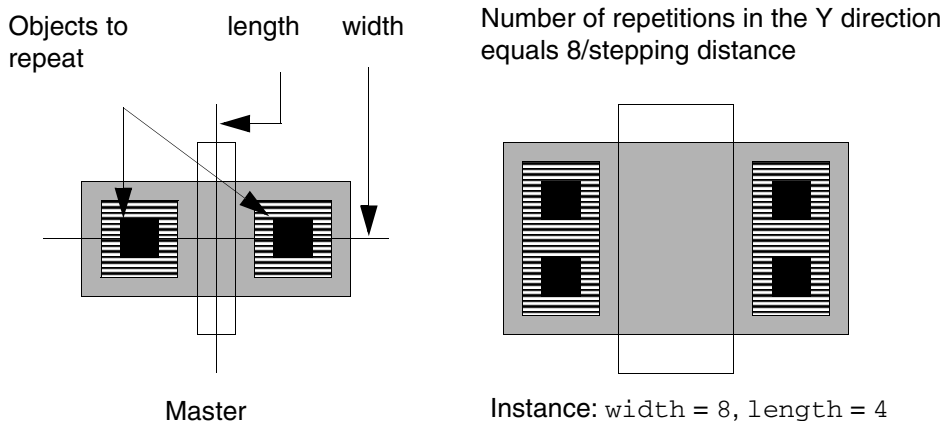


- If you include a group of objects in a repetition set and divide the set by a stretch control line, stretching takes place before the objects are repeated. Even though the number of repetitions is defined as a function of the width (width/stepping distance), because stretching takes place first, it is possible to repeat contacts on the source and drain of the

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

transistor while varying both the channel length and width. The value of the gate width after the stretch is used to compute the number of repetitions.



Stretching Objects in Repetition Groups

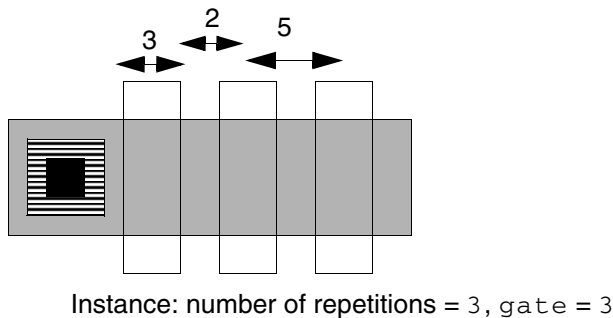
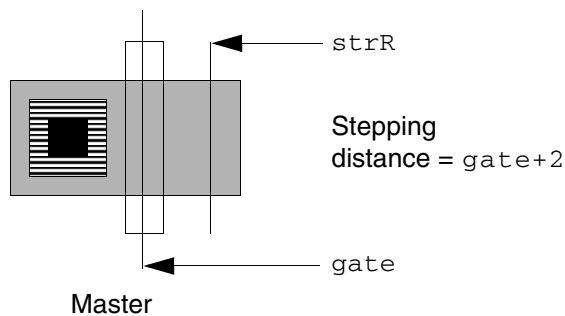
Objects in a repetition group crossed by a stretch control line are not stretched by default. You might want to stretch objects in a repetition group when, for example, you generate multiple transistor gates and want each gate to have a stretchable channel length.

You can stretch objects in a repetition group by turning on *Stretch Horizontally Repeated Figures* in the Stretch in X Form or by turning on *Stretch Vertically Repeated Figures* in the Stretch in Y Form. In the following master, the gate length is 2 and the stepping distance is 4 (gate length of 2 plus gate-to-gate spacing of 2). If the stretch control line for the gate

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

length is called `gate`, the stepping distance of these gates can be specified as `gate+2`. The stretch control line `strR` is designated a dependent stretch control line.



Using Dependent Stretch Control Lines

By default, stretching takes place before repetition. However, you can set the stretch control line dependent on the repetition parameter.

- When the stretch parameter is primary (default), the amount of stretch controls the number of repetitions.
- When the repetition parameter is primary (stretch control line is dependent), the number of repetitions controls the amount of stretch.

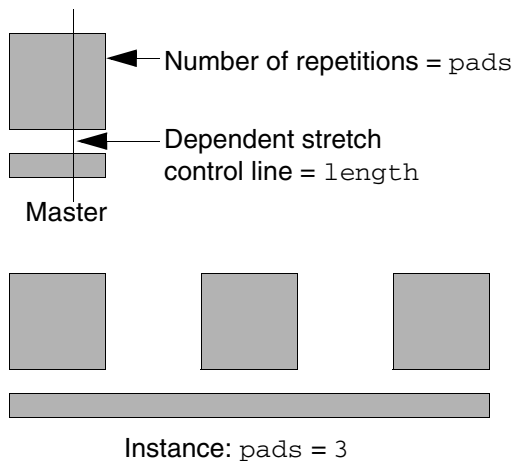
You can specify the value for the primary parameter, but you cannot control the secondary parameter. The system uses the value you specify for the primary parameter to compute the value for the secondary parameter. You designate which parameter is primary and which is secondary depending on which value you want to control.

For example, you want to create a power bus with the maximum number of pads whose stepping distance is 40. You can specify that the primary parameter be either the number of pads (repetition parameter) or the length of the bus (stretch parameter).

Virtuoso Parameterized Cell Reference

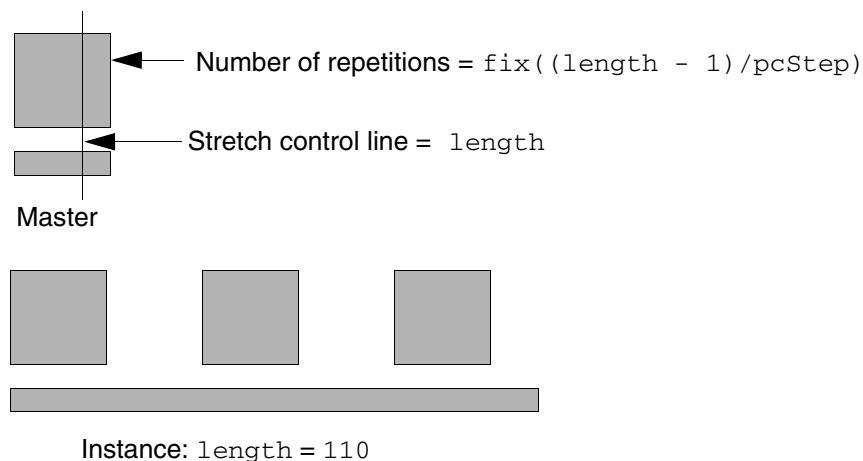
Creating Graphical Parameterized Cells

When the repetition parameter is primary, you set the stretch control line to be dependent. After you define the expressions that control the stepping distance and number of repetitions, you specify the name of a dependent stretch control line in the Stretch in X Form. The dependent stretch control line takes its value from the reference dimension default established in the Stretch in X form and any adjustments to stretch entered in the Repeat in X Form. The amount of stretch is equal to the distance added by the repetitions, as shown in the following example.



When the stretch parameter is primary, the number of repetitions is defined as a function of the stretch parameter and the resulting configuration allows for rounding. If the length of the bus (stretch parameter) is the primary parameter and is not evenly divisible by the stepping distance (40), the stretched power bus can extend beyond the last repeated pad.

If you specify the length to be 110, the number of repetitions is defined as $110/40$. The system rounds the number of repetitions down to 3, and the power bus extends past the last repeated pad by 10.



For more information about using stretch parameters with repetition parameters, refer to [Repetition Commands](#).

Using Stretch with Conditional Inclusion

You can use stretch parameters with conditional inclusion to

- Create conditional stretch control lines
- Include or exclude objects depending on the amount of stretch
- Stretch objects depending on the inclusion or exclusion of other objects

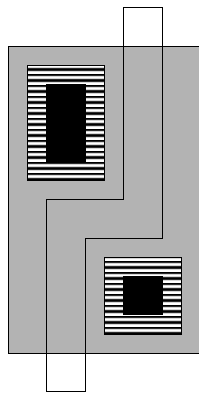
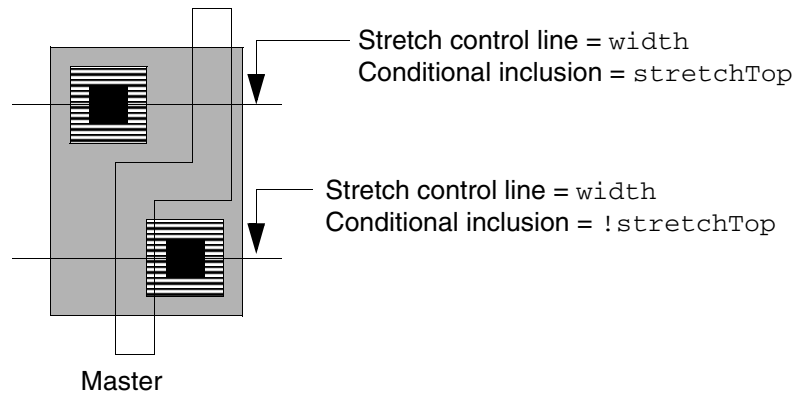
Using conditional inclusion parameters, you can assign two stretch control lines with the same name but with different conditions. For example, you can define two conditionally included stretch control lines with the name `width`. To control where the objects are stretched, you define the conditional inclusion parameter for one of the lines as `stretchTop` and define the inverse condition `!stretchTop` for the other.

When `stretchTop` evaluates to `true`, only the top stretch control line is used. When `stretchTop` evaluates to `false`, only the bottom stretch control line is used. Because these

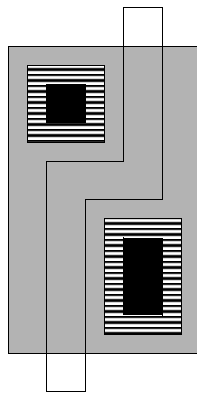
Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

expressions are the inverse of each other, you can toggle between the two when you place an instance of the cell.



Instance: stretchTop = true



Instance: stretchTop = false

For more information about using conditional inclusion parameters with stretch parameters, refer to [Conditional Inclusion Commands](#).

Stretch Menu

Stretch commands set a parameter that stretches objects. Use the *Stretch* commands to stretch objects in the X direction, Y direction, or both directions. Use *Qualify* to include or exclude objects from the stretch. Use *Modify* to change the stretch control line values. Use *Redefine* to redraw and change parameters for an existing stretch control line.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

Stretch in X

Defines how to stretch the objects horizontally. Objects intersected by the stretch control line are stretched horizontally at the point of intersection. Objects to the left, right, or left and right of the stretch control line are moved horizontally.

To use Stretch in X,

1. In a layout window, choose *Pcell – Stretch – Stretch in X*.

The system prompts you to draw a vertical line to control the horizontal stretch. Refer to the [control line rules](#) for more information.

2. Draw the control line.
3. Double-click, or press `Return`, to enter the last vertex.

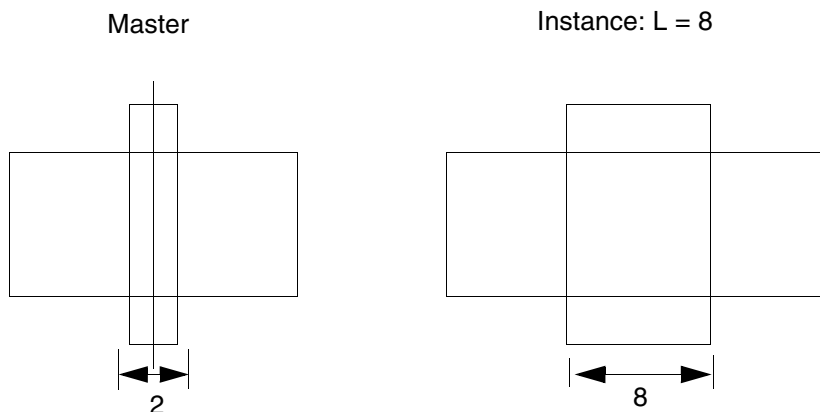
The [Stretch in X form](#) appears.

4. In the *Name or Expression for Stretch* field, type a variable name or expression.

If you assign a variable name, you are prompted to give a value for this name when you place the Pcell. If you assign a SKILL expression, the Pcell program calculates the value of the expression using the values for the SKILL variables.

5. Set the options you want used for the stretch.
6. Click *OK* and press `Escape` to end the command.

Stretch in X Example



Name or Expression for Stretch = L
Stretch Direction = right
Reference Dimension = 2

Stretch in X SKILL Function

```
pcHIDefineStretch( "right" ) => t | nil
```

Stretch in Y

Defines how to stretch objects in a cellview in the vertical direction. Objects intersected by the stretch control line are stretched vertically at the point of intersection. Objects above, below, or above and below the stretch control line are moved vertically.

To use Stretch in Y,

1. In a layout window, choose *Pcell – Stretch – Stretch in Y*.

The system prompts you to draw a horizontal line to control the vertical stretch. Refer to the [control line rules](#) for more information.

2. Draw the control line.
3. Double-click, or press `Return`, to enter the last vertex.

The [Stretch in Y form](#) appears.

4. In the *Name or Expression for Stretch* field, type a variable name or expression.

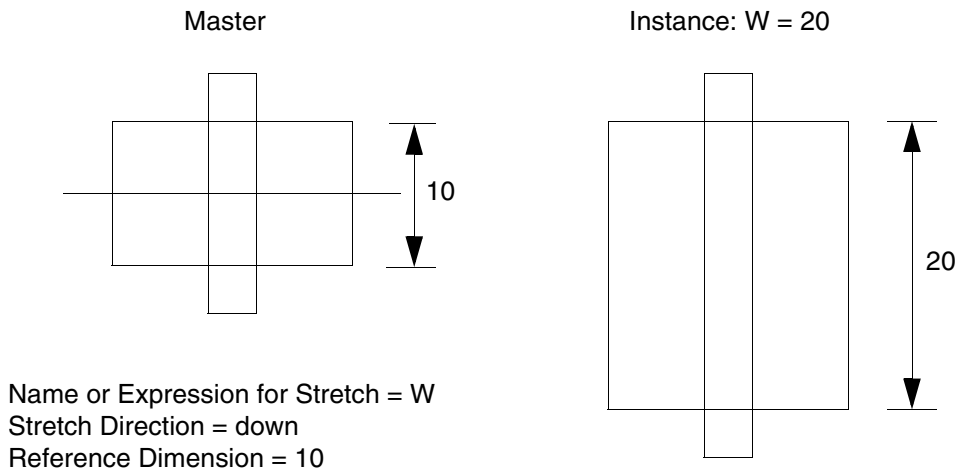
If you assign a variable name, you are prompted to give a value for this name when you place the Pcell. If you assign a SKILL expression, the Pcell program calculates the value of the expression using the values given for the variables in it.

5. Set the options you want used for the stretch.
6. Click *OK* and press `Escape` to end the command.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

Stretch in Y Example



Stretch in Y SKILL Function

```
pcHIDefineStretch( "up" ) => t | nil
```

Qualify

Specifies which objects are affected by a stretch control line. By default, all objects not in repetition groups are affected by the stretch.

To use Qualify,

1. In a layout window, choose *Pcell – Stretch – Qualify*.

The system prompts you to select a stretch control line.

2. Click the stretch control line you want to qualify.

The program highlights the stretch control line and prompts you to choose the objects you want affected by the stretch.

3. Click to select objects you want affected by the stretch control line.

Each object is highlighted as you click it. Once an object is selected, you can deselect it by pressing `Control` and clicking on it again.

4. Double-click, or press `Return`, to stop selecting objects.

All selected objects are affected by the stretch. The stretch parameter is now listed as a qualified stretch in the Summarize window.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

When all objects are qualified by selecting them, all objects are affected by the stretch control line. When no objects are qualified by selecting them, the default is applied, and all objects not in repetition groups are affected by the stretch control line.

Verifying Qualify

1. In a layout window, choose *Pcell – Stretch – Qualify*.

The system prompts you to select a stretch control line.

2. Click a stretch control line.

The system highlights the stretch control line and the qualified objects.

3. Do one of the following:

- ☐ If the objects you want qualified are highlighted, double-click or press `Return`.
- ☐ Press `Escape` to end this command.
- ☐ If you want to change the qualified objects, click any object to qualify it or press `Control` and click any selected object to unqualify it.

The system highlights selected objects and unhighlights deselected objects.

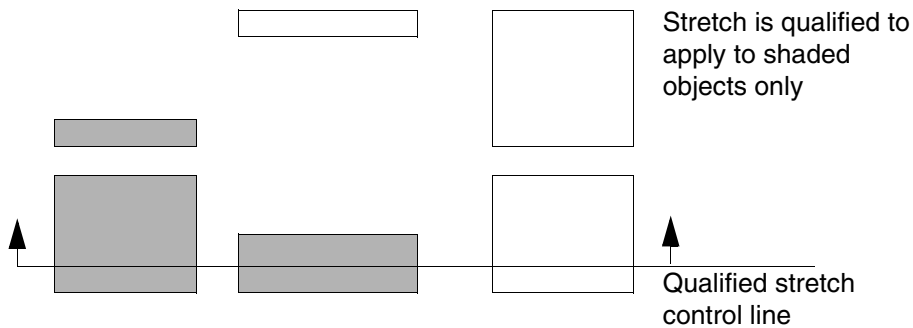
4. Double-click, or press `Return`, to stop selecting objects.

Virtuoso Parameterized Cell Reference

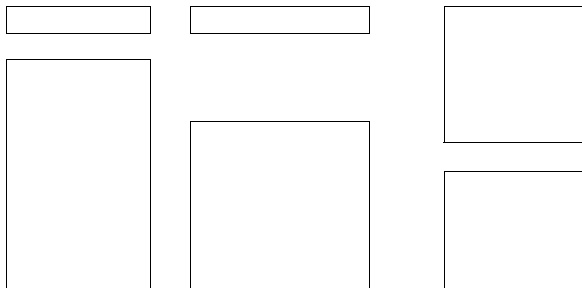
Creating Graphical Parameterized Cells

Qualify Example

Master



Instance



Effect of qualified stretch

Qualify SKILL Function

```
pcHIQualifyStretchLine( ) => t | nil
```

Modify

Modifies the parameters for a stretch control line.

To use Modify,

1. In a layout window, choose *Pcell – Stretch – Modify*.

The system prompts you to select the stretch control line you want to modify.

2. Click the stretch control line you want to modify.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

Either the Stretch in X Form or the Stretch in Y Form appears, depending on whether you are modifying an X stretch line or a Y stretch line.

3. Do one of the following:

- ☐ If you do not want to change any of the stretch parameters, click *Cancel*.
- ☐ Change the settings on the form to indicate changes to the stretch parameter and click *OK*.

The system assigns the new parameters to the stretch control line.

Modify Stretch SKILL Function

```
pcHIModifyStretchLine( ) => t | nil
```

Redefine

Lets you redefine a previously defined stretch control line or change the parameters assigned to a stretch control line.

To use Redefine,

1. In a layout window, choose *Pcell – Stretch – Redefine*.

The system prompts you to select a stretch control line.

2. Click the stretch control line you want to redefine.

The system prompts you to draw a new stretch control line. If you want to change the parameter values without changing the line itself, double-click, or press `Return`. Either the Stretch in X Form or the Stretch in Y Form appears, depending on whether you are redefining an X stretch line or a Y stretch line. Go to step 7.

3. Click the beginning of the new line.

4. Click each vertex of the line.

5. Double-click, or press `Return`, on the last vertex.

Either the Stretch in X Form or the Stretch in Y Form appears, depending on whether you are redefining an X stretch line or a Y stretch line.

6. If you do not want to change any of the parameter values, click *OK*.

The system deletes the old stretch control line.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

7. Change the settings on the form to indicate changes to the stretch parameters and click *OK*.

The system assigns the new parameters to the stretch control line.

Redefine Stretch SKILL Function

```
pcHIRedefineStretchLine( ) => t | nil
```

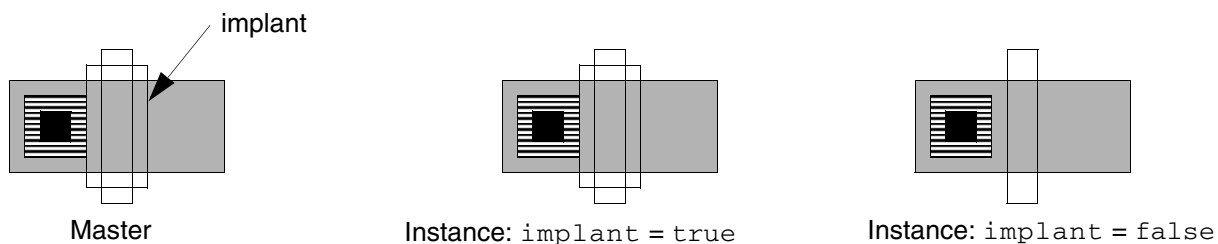
Conditional Inclusion Commands

A conditional inclusion parameter lets you include or exclude objects from a cellview when you place an instance of a Pcell.

- Including or Excluding Objects
- Using Conditional Stretch Control Lines
- Using Conditional Inclusion with Repetition
- Using the *Conditional Inclusion* menu, including
 - ☐ Grouping objects to be conditionally included (see “Define” on page 76)
 - ☐ Including or excluding objects from conditional inclusion groups or changing the definition of conditional inclusion parameters (see “Modify” on page 78)
 - ☐ Removing conditional inclusion parameters from groups of objects (see “Delete” on page 79)
 - ☐ Displaying parameter information about conditional inclusion groups (see “Show” on page 79)

Including or Excluding Objects

To assign a conditional inclusion parameter, you must assign a parameter definition, or conditional expression, to the objects to be included. The parameter definition must be a Cadence® SKILL language expression.



If this SKILL expression is as simple as `implant`, a button on the Create Instance form prompts you to include or exclude the implant when you place the Pcell. To include the implant, turn on the *implant* button.

If the parameter definition is a SKILL expression based on another parameter of the Pcell, such as `fix(numgates) == 1`, the system evaluates this expression when you place an

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

instance. In this case, the expression for the conditional inclusion is not shown on the Create Instance form because you are not required to enter a value.

- If the expression evaluates to true (`t`), the system includes the objects in the instance.
- If the expression evaluates to false (`nil`), the system does not include the objects in the instance.

You can associate an object or group of objects with multiple conditions so that all conditions must evaluate to true for the objects to be included.

Using Conditional Stretch Control Lines

You can define a stretch control line as conditional. If the conditional parameter of the stretch control line evaluates to false, the object becomes the size of its reference dimension, regardless of the value of its parameter.

You can use conditional inclusion parameters with stretch parameters to

- Create conditional stretch control lines
- Include or exclude objects depending on the stretch
- Stretch objects depending on the inclusion or exclusion of other objects

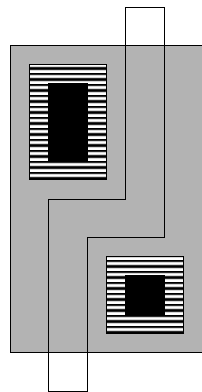
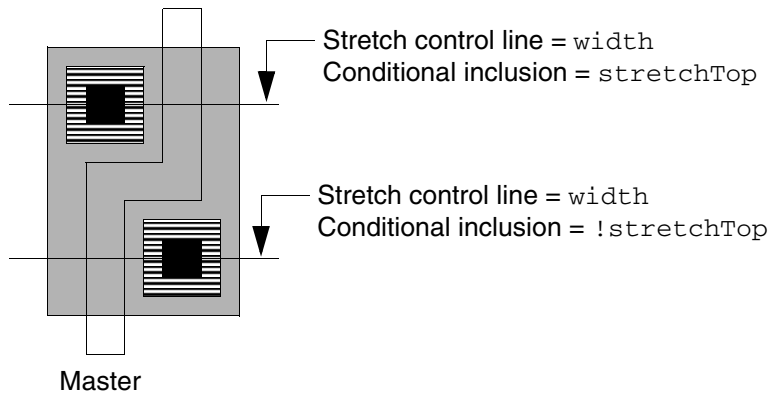
Using conditional inclusion parameters, you can assign two stretch control lines with the same name but with different conditions. For example, you can define two conditionally included stretch control lines with the name `width`. To control where the objects are stretched, you define the conditional inclusion parameter for one of the lines as `stretchTop` and define the inverse expression `!stretchTop` for the other.

When `stretchTop` evaluates to true, only the top stretch control line is used. When `stretchTop` evaluates to false, only the bottom stretch control line is used. Because these

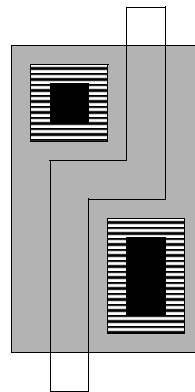
Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

expressions are the inverse of each other, you can toggle between the two when you place an instance of the cell.



Instance: stretchTop = true



Instance: stretchTop = false

You can use dependent stretch control lines to stretch or compress related objects to overlap or avoid the included objects. The inclusion or exclusion of the selected objects determines the value of the dependent stretch control line.

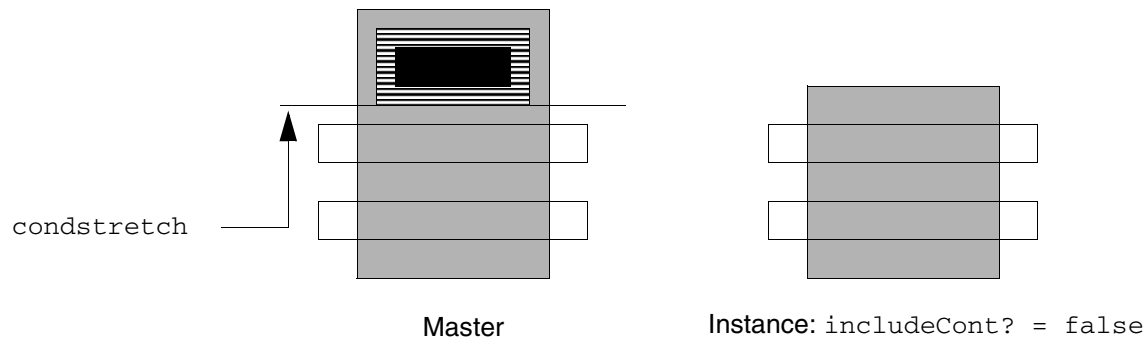
For example, you define a stretch control line with the name `condstretch` that compresses the diffusion when the contact is not included in the instance. Create a conditional inclusion group to include the contact and its surrounding metal. Define the conditional inclusion parameter as `includeCont?` and designate `condstretch` as a dependent stretch control line.

The value for the adjustment to stretch is the amount to stretch when the conditional inclusion evaluates to false. If you want the stretch to compress the object when the conditional objects are excluded, you either choose *down* as the stretch direction or give a negative number for the value of the stretch. When you place an instance of the Pcell, a button prompts you to

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

choose `true` or `false` for the `includeCont?` parameter. If the contact is not included, the enclosed area is compressed by the amount specified in the adjustment to stretch.



The Virtuoso[®] Parameterized Cell (Pcell) program treats the dependent stretch control line as a secondary parameter of the Pcell. A secondary parameter takes its value from the reference dimension and any offsets caused by the exclusion of conditional objects.

For more information about using conditional inclusion parameters with stretch parameters, refer to [Stretch Commands](#).

Using Conditional Inclusion with Repetition

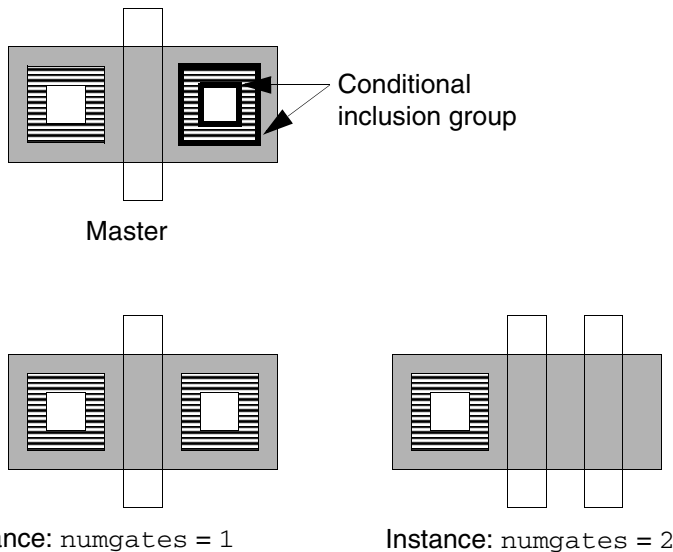
You can assign both conditional inclusion parameters and repetition parameters to a Pcell. The conditional object is included only if the conditional expression, which is dependent on the repetition parameter, evaluates to `true`.

If you reference the same SKILL symbol in both a repetition parameter and a conditional expression, the parameter type is `float`. In this case, the default parameter type for the conditional inclusion is not Boolean.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

For example, if you want a contact and its metal to be included only when a single gate is specified, you define the conditional inclusion parameter as `fix(numgates) == 1`. The conditional contact and metal are included only when `numgates` equals 1.



For more information about using conditional inclusion parameters with repetition parameters, refer to [Repetition Commands](#).

Conditional Inclusion Menu

Conditional Inclusion commands set a parameter that includes or excludes objects depending on the conditions you set. These commands can be used in conjunction with stretch or repetition commands. Use *Define* to group objects to be conditionally included. Use *Modify* to change the conditional inclusion parameters assigned to the Pcell. Use *Delete* to remove a conditional inclusion parameter from an object or group of objects. Use *Show* to highlight each conditional inclusion group separately and display information about the group.

Define

Assigns a conditional inclusion parameter to a selected group of objects.

To use Define,

You can preselect objects or select them after starting the command.

1. In a layout window, choose *Pcell – Conditional Inclusion – Define*.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

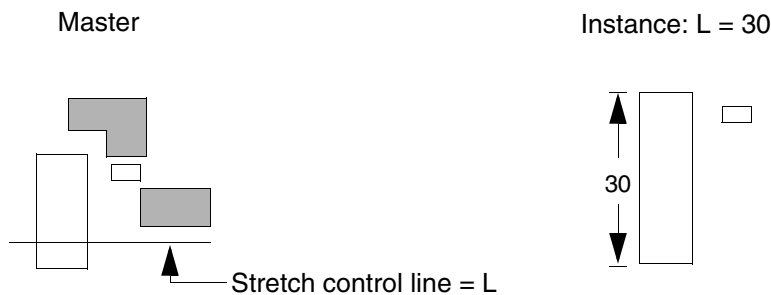
If you have not preselected any objects, the system prompts you to select the objects you want to include in the conditional group.

2. Click the objects you want included.
3. Double-click, or press `Return`, to stop selecting objects.

The Conditional Inclusion form appears.

4. In the *Name or Expression* field, type a definition for the conditional inclusion.
5. (Optional) To set a dependent stretch control line or an adjustment to stretch,
 - ☐ In the *Dependent Stretch* field, type the name of a previously defined stretch control line.
 - ☐ In the *Adjustment to Stretch* field, type the amount to adjust the stretch.
6. Click *OK* and press `Escape` to end the command.

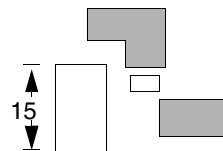
Conditional Inclusion Example



The shaded objects are defined as conditionally included with the following value:

Parameter Name or
Conditional Expression = $L < 20$

Instance: L = 15



Define Condition SKILL Function

```
pcHIDefineCondition( ) => t | nil
```

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

Modify

Lets you change the objects associated with a conditional inclusion or change the expression controlling a conditional inclusion.

To use Modify,

1. In a layout window, choose *Pcell – Conditional Inclusion – Modify*.

The system prompts you to select any object in the conditional inclusion group whose definition you want to modify.

2. Click an object in the conditional inclusion group.

The system highlights all the objects in that conditional inclusion group and prompts you to select objects you want to add to or remove.

3. Include or exclude objects from the conditional inclusion group.

- ☐ To exclude an object from the conditional inclusion group, click the object so it is no longer selected.

- ☐ To include an object in the group, click it so that it is selected.

4. Double-click, or press `Return`, to stop selecting.

The Modify Conditional Inclusion form appears.

The values in this form were defined using the Conditional Inclusion form.

5. Do one of the following:

- ☐ To keep the original values for the conditional inclusion so that only the objects in the repetition group are modified, click *Cancel*.

- ☐ To enter new values, fill in the form and click *OK*.

If the object you selected in step 2 is associated with more than one conditional inclusion group, the system highlights the next conditional inclusion group and prompts you to point to objects you want to add to or remove from that group.

6. Press `Escape` to end the command.

Modify Condition SKILL Function

```
pcHIModifyCondition( ) => t | nil
```

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

Delete

Deletes a conditional inclusion definition. Does not delete the objects in the conditional inclusion group.

To use Delete,

1. In a layout window, choose *Pcell – Conditional Inclusion – Delete*.

The system prompts you to select an object in the conditional inclusion group whose definition you want to delete.

2. Click any object in the group.

The system highlights all the objects in the conditional inclusion group and displays the Delete Conditional Inclusion form, showing information about the group. These fields are grayed out, and you cannot modify the data here.

If there is more than one conditional inclusion group, the window displays information about only one of the groups.

The values in this form were defined using the Conditional Inclusion form.

3. Click the option you want:

- ☐ *OK* closes the form and deletes the conditional inclusion group.
- ☐ *Cancel* closes the form and does not delete the conditional inclusion group.
- ☐ *Apply* deletes the conditional inclusion group and lists the next conditional inclusion group on the form.
- ☐ *Next* saves the first conditional inclusion group and displays the next conditional inclusion group.

Delete Condition SKILL Function

```
pcHIDeleteCondition( ) => t | nil
```

Show

Highlights objects in a conditional inclusion group and displays information about the group.

To use Show,

1. In a layout window, choose *Pcell – Conditional Inclusion – Show*.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

The system highlights the objects in the conditional group and displays the Show Conditional Inclusion window, showing information about the group.

2. Do one of the following:

- ☐ If there is more than one conditional group, click *OK* to view the next group. When you are finished viewing, close the window by clicking *Cancel*.
- ☐ If there is only one conditional group, close the window by clicking *Cancel*.

Show Condition SKILL Function

```
pcHIDisplayCondition( ) => t | nil
```


Repetition Commands

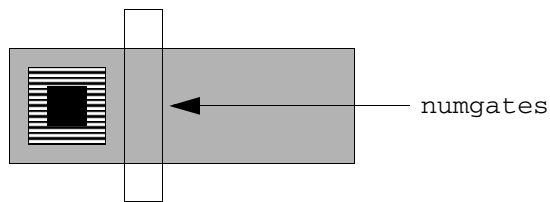
A repetition parameter arrays or repeats objects in the X direction, Y direction, or both directions. This chapter discusses three examples of the use of repetition parameters. You can use repetition parameters for

- [Specifying the Direction in which Objects Repeat](#)
- [Specifying the Number of Repeated Objects](#)
- [Specifying the Stepping Distance](#)
- [Specifying Parameter Definitions](#)
- [Repeating Pins and Terminals](#)
- [Using Repetition with Stretch](#)
- [Using Repetition with Conditional Inclusion](#)
- Using the *Repetition* menu, including
 - Repeating objects in a horizontal direction (see [“Repeat in X”](#) on page 94)
 - Repeating objects in a vertical direction (see [“Repeat in Y”](#) on page 95)
 - Repeating objects in both horizontal and vertical directions to create a two-dimensional array (see [“Repeat in X and Y”](#) on page 96)
 - Changing the objects included in a repetition group or the parameters assigned to the group (see [“Modify”](#) on page 97)
 - Removing the repetition parameter assigned to a group of objects (see [“Delete”](#) on page 99)
 - Displaying the parameter information about all objects in a repetition group (see [“Show”](#) on page 99)

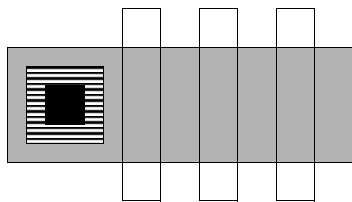
Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

The following example shows a repetition parameter named `numgates` that creates a series of gates.



Master



Instance: `numgates = 3`

Specifying the Direction in which Objects Repeat

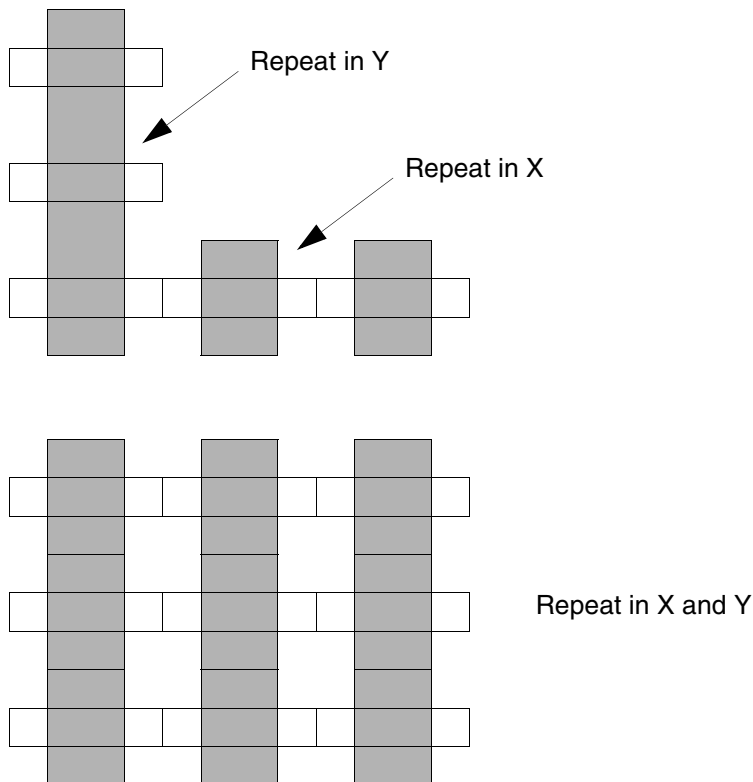
You can specify the direction in which objects repeat by using one of the following commands:

- *Repeat in X*, which defines objects to be repeated horizontally
- *Repeat in Y*, which defines objects to be repeated vertically
- *Repeat in X and Y*, which defines objects to be repeated both horizontally and vertically

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

If you want to create an L-shaped repetition, use the *Repeat in X* and *Repeat in Y* commands separately to repeat an object horizontally and vertically. The *Repeat in X and Y* command creates a two-dimensional array.



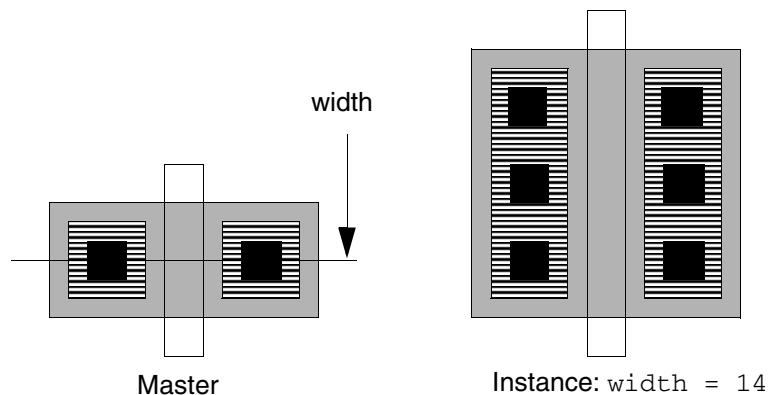
When you place an instance of a Pcell whose repetition parameters are defined with names, such as `gatesX` and `gatesY`, the Create Instance form lists one parameter name for the number of repetitions in the X direction and another parameter name for the number of repetitions in the Y direction. These names look the same whether you use *Repeat in X* and *Repeat in Y* or *Repeat in X and Y*. If you expect a two-dimensional array but the placed Pcell is an L shape, you used the *Repeat in X* and *Repeat in Y* commands instead of the *Repeat in X and Y* command when you created the master.

If the Cadence® SKILL language expression controlling the number of repetitions evaluates to less than 1, the objects are not repeated. If the expression evaluates to other than a whole number, the program rounds the number down to the nearest integer.

You can create a two-dimensional array of all objects in a Pcell by using the *Rows, Columns* options on the Create Instance form instead of the *Repeat in X and Y* parameter.

Specifying the Number of Repeated Objects

Use a SKILL expression to specify the number of repetitions. This expression can contain references to other parameters of the Pcell. For example, the number of contacts used in strapping the source and drain regions of a transistor can be an expression that is dependent on the stretch parameter. If the stretch parameter is named `width` and the stepping distance (pitch) for the 2x2 contacts is 4, the number of repetitions can be `width` divided by 4. The number of repetitions is rounded down to a whole integer.



Any symbol you use in the SKILL expression is automatically recorded as a parameter of the Pcell, unless it is one of the reserved symbols described in [Table 6-1](#) on page 86. For example, you might use the expression `numgates` for the number of repetitions parameter. When you place an instance of the Pcell, the editor prompts you to type in a value for the parameter `numgates`.

If you place an instance of a Pcell with repetition parameters whose values for repetition in the X and Y directions are greater than 1, but the placed Pcell shows only one object in either or both directions, check the stepping distance in the [master](#). If you did not specify a stepping distance, the default is 0 and the repetitions in the placed Pcell are placed on top of each other.

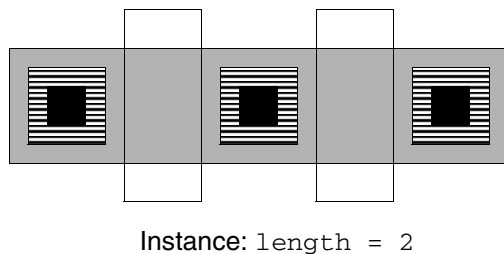
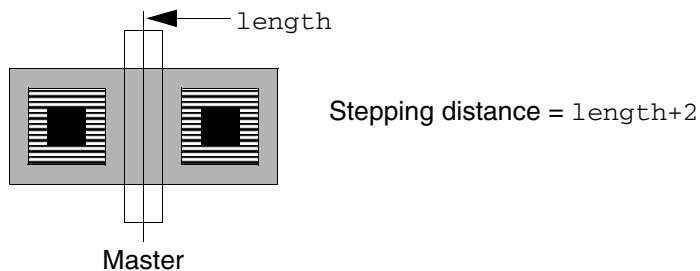
Specifying the Stepping Distance

Use a name or SKILL expression to specify stepping distance. The stepping distance of the repeated objects is the centerline-to-centerline distance, or pitch. This expression can contain references to other parameters of the Pcell. For example, the stepping distance of contacts used in strapping the source and drain regions of a transistor can be an expression that is

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

dependent on the stretch parameter named `length`. If the minimum spacing to the gate region is 1, the stepping distance for contacts can be `length+2`.



By default, repetition takes place in a positive direction (upward or to the right). To repeat in a negative direction (downward or to the left), you must use a negative number for the stepping distance.

Specifying Parameter Definitions

Parameter definitions for repetition parameters and stepping distance parameters are usually SKILL expressions rather than names. The default type for the stepping distance is `float`.

If you specify a conditional expression in a parameter definition that references either the repetition or stepping distance parameters, the type of operands you use is important to compare values in SKILL. For example, if you use the parameter `numgates` to control the number of repetitions, and you specify a conditional inclusion that depends on `numgates` having a value of 2, you must define the conditional inclusion parameter as either

```
numgates == 2.0
```

or

```
fix (numgates) == 2
```

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

Reserved symbol names are used in expressions that control the stepping distance and number of repetitions. The following table lists the reserved symbol names.

Table 6-1 Reserved Symbol Names

Symbol	Meaning
<code>pcStep</code>	Stepping distance (one-dimensional repetition)
<code>pcRepeat</code>	Number of repetitions (one-dimensional repetition)
<code>pcStepX</code>	Horizontal stepping distance
<code>pcStepY</code>	Vertical stepping distance
<code>pcRepeatX</code>	Number of horizontal repetitions
<code>pcRepeatY</code>	Number of vertical repetitions
<code>pcIndexX</code>	Loop control variable for horizontal repetition (0 - <code>fix(pcRepeatX) - 1</code>)
<code>pcIndexY</code>	Loop control variable for vertical repetition (0 - <code>fix(pcRepeatY) - 1</code>)

You can use either `pcRepeat` in the expression for the stepping distance or `pcStep` in the expression for the number of repetitions, but not both.

When you specify a repetition in both X and Y, you cannot use `pcStep` or `pcRepeat`. You must use `pcStepX`, `pcStepY`, `pcRepeatX`, or `pcRepeatY`. The Virtuoso® Parameterized Cell (Pcell) program checks for this and warns you in the CIW if you type the wrong symbol name in the Repeat in X and Y form.

Repeating Pins and Terminals

When you include a pin in the set of objects to repeat, the Pcell compiler does one of the following:

- Treats each repeated pin as another pin of the single terminal to which the original pin belongs (default)
- Treats each repeated pin as if it belongs to a different terminal

To treat each repeated pin as if it belongs to a different terminal, add a Boolean-valued property to the pin shape using the Edit – Properties command. The name of the property is `pcMultiTerm`, and you must set its value to `true`.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

If the pin is repeated in both X and Y directions, you can associate each pin in the X direction or the Y direction with individual terminals using the `pcMultiTermX` and `pcMultiTermY` properties. If you use these properties, do not use the `pcMultiTerm` property.

- If you want each pin repeated in the horizontal direction to be associated with a different terminal, add the `pcMultiTermX` property to each pin figure and set the property value to `true`.
- If you want each pin repeated in the vertical direction to be associated with a different terminal, add the `pcMultiTermY` property to each pin figure and set the property value to `true`.

Note: Setting both `pcMultiTermX` and `pcMultiTermY` to `true` is equivalent to setting `pcMultiTerm` to `true`.

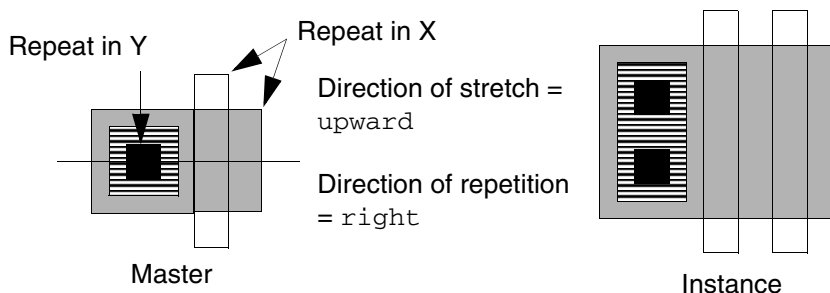
Using Repetition with Stretch

You can combine stretch parameters with repetition parameters in the same Pcell. By default, stretching takes place before repetition. If you define the stretch control line to be dependent, repetition takes place before stretching.

Using Stretch Parameters with Repeated Objects

When a stretch control line crosses an object to be repeated, the effect depends on the stretch direction of the stretch control line and the direction of repetition of the object.

- If the stretch direction is perpendicular to the repetition direction, the object is stretched before being repeated.

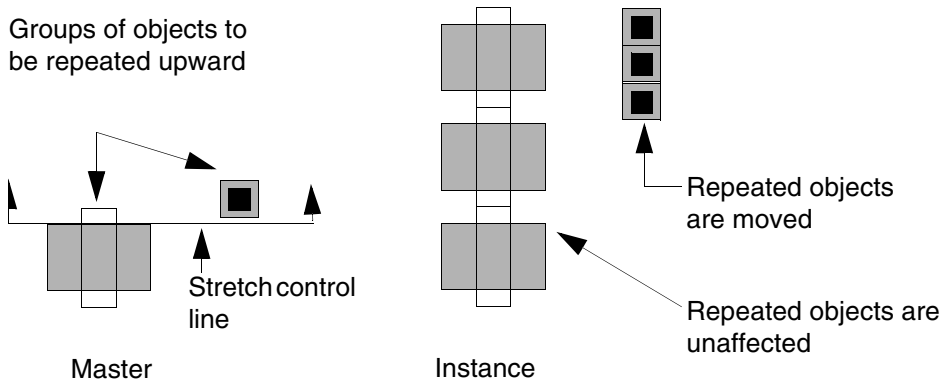


- If the stretch direction is parallel to the direction of repetition, the object or group of objects is not affected by that stretch line. The repetition group is affected by the stretch

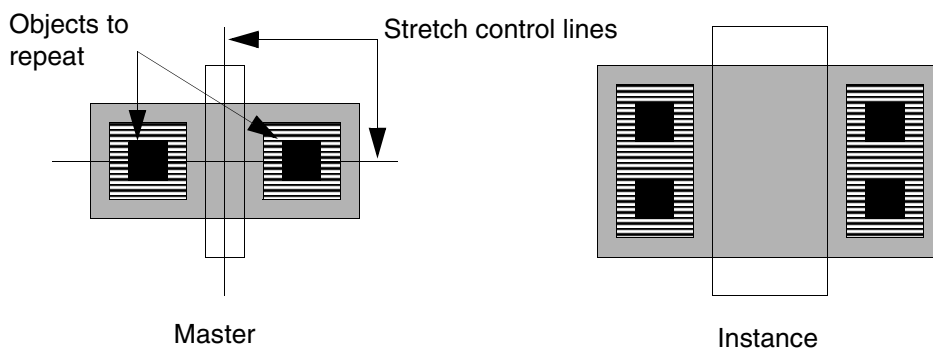
Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

only if all objects in the group lie on the same side of a stretch control line that stretches in the same direction as the repetition.



By default, stretching takes place before repetition. If you include a group of objects in the same repetition set and divide the set by a stretch control line, the stretching on that stretch control line takes place before the objects are repeated. The number of repetitions can be defined as a function of the width (width/stepping distance). Because stretching takes place first, it is possible to repeat contacts on the source and drain of the transistor while varying both the channel length and width. The number of repetitions is computed using the value for the gate width after the stretch.



Stretching Objects in Repetition Groups

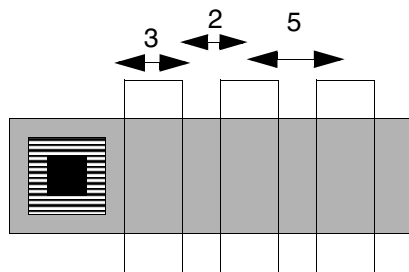
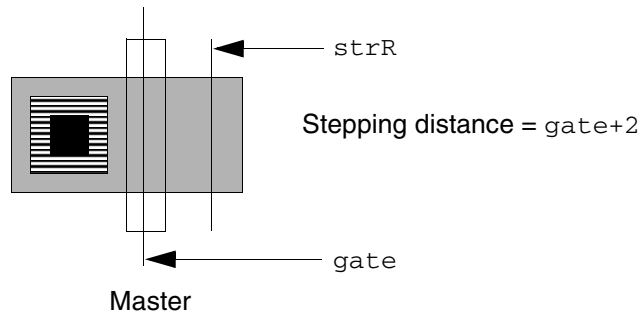
Objects that are in a repetition group and cross a stretch control line, by default, are not stretched. There are times you might want to stretch objects in a repetition group, such as when you generate multiple transistor gates. In this case, you want each gate to have a stretchable channel length.

You can stretch objects in a repetition group by selecting *true* for *Stretch Horizontally Repeated Figures* in the *Stretch in Y Form* or turning on *Stretch Vertically Repeated Figures* in the *Stretch in Y Form*. In the following master, the gate length is 2 and the stepping

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

distance is 4 (gate length of 2 plus gate-to-gate spacing of 2). If the stretch control line for the gate length is called `gate`, the stepping distance of these gates can be specified as `gate+2`. The stretch control line `strR` is designated as a dependent stretch control line.



Instance: number of repetitions = 3,
gate = 3

Using Dependent Stretch Control Lines

By default, stretching takes place before repetition. However, you can designate the stretch control line as dependent on the repetition parameter.

- When the stretch parameter is primary (default), the amount of stretch controls the number of repetitions.
- When the repetition parameter is primary (stretch control line is dependent), the number of repetitions controls the amount of stretch.

You can specify the value for the primary parameter, but you cannot control the secondary parameter. The system uses the value you specify for the primary parameter to compute the value for the secondary parameter. You designate which parameter is primary and which is secondary, depending on which parameter you want to control.

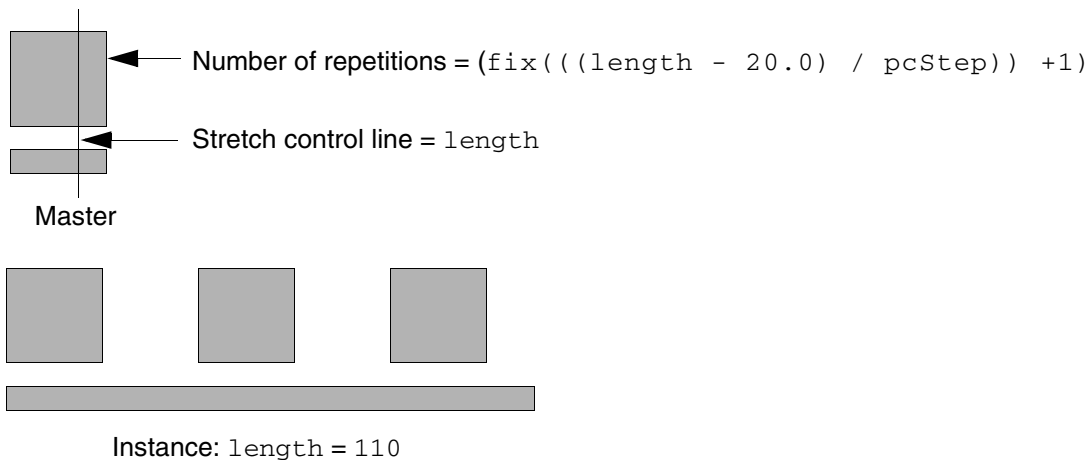
For example, you want to create a power bus with the maximum number of pads whose stepping distance is 40. You can specify the primary parameter as the length of the bus (stretch parameter) or the number of pads (repetition parameter).

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

- When the stretch parameter is primary, the number of repetitions is defined as a function of the stretch parameter and the resulting configuration allows for rounding. If the length of the bus (stretch parameter) is the primary parameter and is not evenly divisible by the stepping distance (40), the stretched power bus can extend beyond the last repeated pad.

If you specify the length as 110, the number of repetitions is defined as $40/110$. The system rounds the number of repetitions down to 3, and the power bus extends past the last repeated pad by 10.

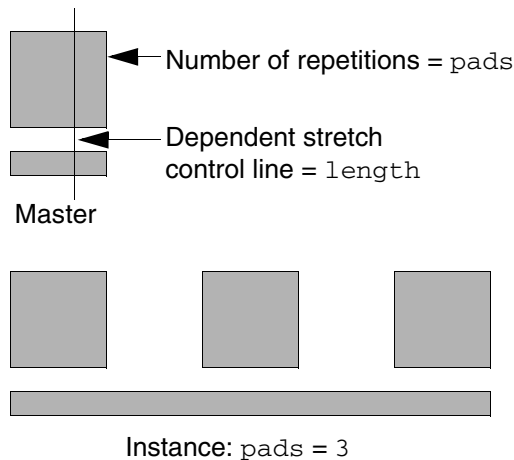


- When the repetition parameter is primary, you set the stretch control line to be dependent. After you define the expressions that control the stepping distance and number of repetitions, you specify the name of a dependent stretch control line in the Repeat in X Form. The dependent stretch control line takes its value from the reference dimension default established in the Stretch in X Form and any adjustments to stretch entered in the Repeat in X form.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

If you specify the number of pads as 3, the Pcell program computes the value of the stretch control line to be 100. The amount of stretch is equal to the distance added by the repetitions, as shown in the following example.



Using an Adjustment to Stretch

You can also enter an adjustment to stretch for the dependent stretch control line. This adjustment is usually a SKILL expression involving both the stepping distance and number of repetitions. The default is

```
fix((pcRepeat - 1) * pcStep
```

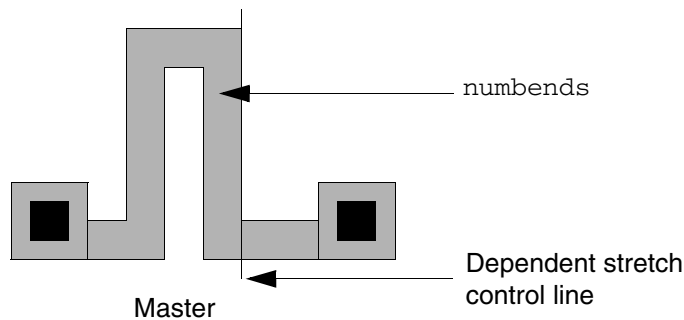
where `fix` is the rounding function, `pcRepeat - 1` is the number of repetitions minus the original repeated object, and `pcStep` is the stepping distance. When you place an instance of the Pcell and specify a number of repetitions, the program uses the values for the stepping distance and the number of repetitions to determine the amount of the stretch.

The serpentine resistor illustrates this concept. The number of repetitions `numbends` is the primary parameter that determines the length of the resistor, and the location of the pins at

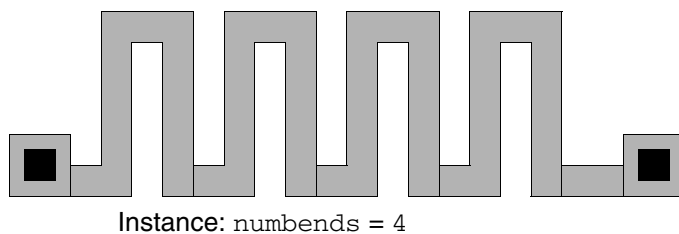
Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

either end of the resistor is determined by a dependent stretch control line. You can enter a custom adjustment to stretch to determine the position of the pins at the end of the resistor.



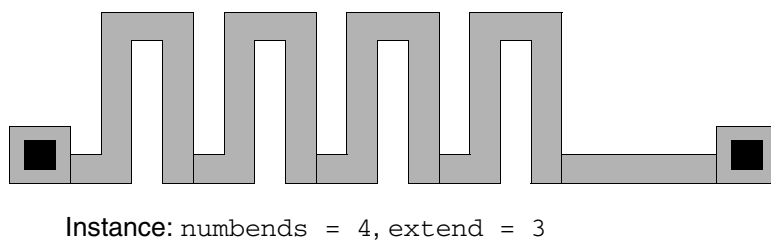
The default adjustment to stretch positions the pins in the same relative position as in the master.



An adjustment to stretch such as

```
fix(((pcRepeat -1) * pcStep) + extend)
```

extends the resistor to the desired length.



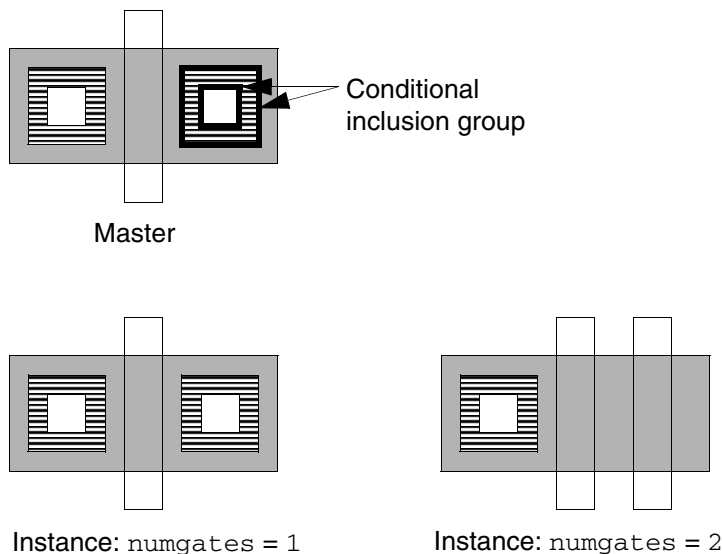
If you define a single stretch control line as a dependent stretch for more than one repetition group, the stretch adjusts for each repetition. Conversely, you can use the same definition for multiple stretch control lines. If you specify that definition as a dependent stretch, the program adjusts all stretch control lines with that definition by the specified amount.

Using Repetition with Conditional Inclusion

You can assign both conditional inclusion and repetition parameters to a Pcell. In this case, the object is included only if the conditional expression, which is dependent on the repetition value, evaluates to `true`.

If you reference the same symbol in both a repetition expression and a conditional expression, the parameter type is `float`. In this case, the default parameter type for the conditional inclusion is not Boolean.

For example, if you want a contact and its metal to be included only when a single gate has been specified, you can define the conditional inclusion parameter as `fix(numgates) == 1`. The conditional contact and metal are included only when `numgates` equals 1.



For more information about using conditional inclusion parameters with repetition parameters, refer to the [Conditional Inclusion Commands](#)

Repetition Menu

Repetition commands set a parameter that repeats the object in the X direction, Y direction, or both. You can use these commands with stretch or conditional inclusion commands. Use the *Repeat* commands to repeat objects in the X direction, Y direction, or both directions. Use *Modify* to change the objects included in a repetition group or the parameters assigned to the group. Use *Delete* to remove the repetition parameters assigned to a group of objects. Use *Show* to highlight each repetition group separately and display information about the group.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

Repeat in X

Defines objects to be repeated horizontally.

To use Repeat in X,

You can preselect objects or select them after starting the command.

1. In a layout window, choose *Pcell – Repetition – Repeat in X*.

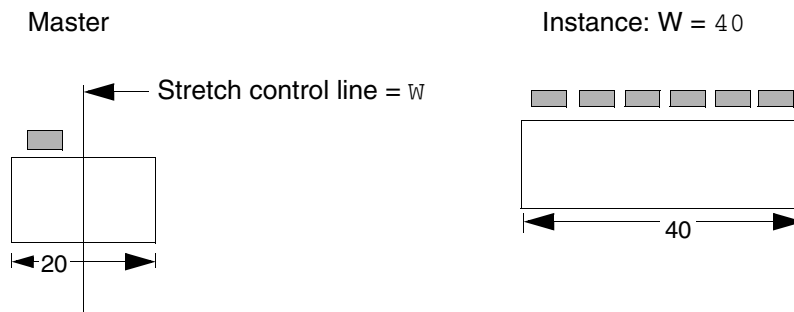
If you did not preselect objects, the program prompts you to select the objects you want to include in the repetition group.

2. Click the objects you want to include in the repetition group.
3. Double-click, or press `Return`, to stop selecting.

The Repeat in X form appears.

4. In the *Stepping Distance* field, type a stepping distance.
5. In the *Number of Repetitions* field, type the number of repetitions.
6. In the *Dependent Stretch* field, type the name of the dependent stretch control line.
7. (Optional) In the *Adjustment Stretch* field, type a value or SKILL expression.
8. Click *OK* and press `Escape` to end the command.

Repeat in X Example



The shaded rectangle is repeated horizontally with the following values:

Stepping Distance = 6.6

Number of Repetitions = $W/pcStep$

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

Repeat in X SKILL Function

```
pcHIDefineRepeat( "horizontal" ) => t | nil
```

Repeat in Y

Defines objects to be repeated vertically.

To use Repeat in Y,

You can preselect objects or select them after starting the command.

1. In a layout window, choose *Pcell – Repetition – Repeat in Y*.

If you did not preselect objects, the program prompts you to select the objects to include in the repetition group.

2. Click the objects you want in the repetition group.
3. Double-click, or press `Return`, to stop selecting.

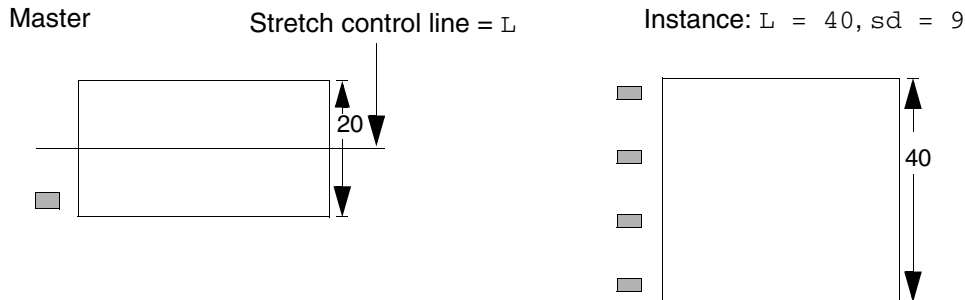
The Repeat in Y form appears.

4. In the *Stepping Distance* field, type a stepping distance.
5. In the *Number of Repetitions* field, type the number of repetitions.
6. In the *Dependent Stretch* field, type the name of the dependent stretch control line.
7. (Optional) In the *Adjustment Stretch* field, type a value or expression.
8. Click *OK* and press `Escape` to end the command.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

Repeat in Y Example



The shaded rectangle is repeated vertically with the following values:

Stepping Distance = sd

Number of Repetitions = L / sd

Repeat in Y SKILL Function

```
pcHIDefineRepeat( "vertical" ) => t | nil
```

Repeat in X and Y

Defines objects to be repeated both horizontally and vertically.

To use Repeat in X and Y,

You can preselect objects or select them after starting the command.

1. In a layout window, choose *Pcell – Repetition – Repeat in X and Y*.

If you did not preselect objects, the program prompts you to select the objects to include in the repetition group.

2. Click the objects you want in the repetition group.
3. Double-click, or press `Return`, to stop selecting.

The Repeat in X and Y form appears.

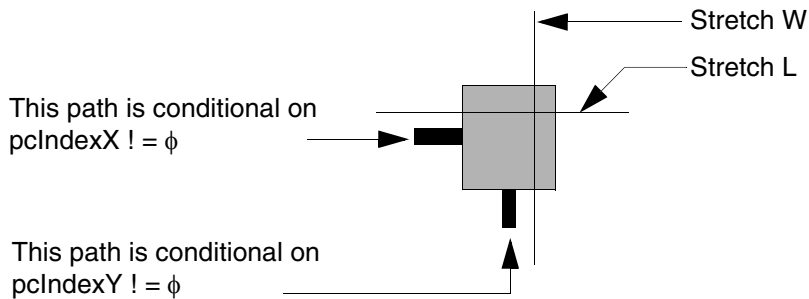
4. In the *Stepping Distance* fields, type stepping distance values for X and Y.
5. In the *Number of Repetitions* fields, type the number of repetitions for X and Y.
6. (Optional) In the *Dependent Stretch* fields, type the names of the dependent stretches for X and Y.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

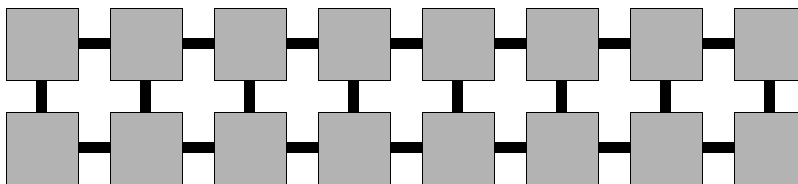
7. (Optional) In the *Adjustment to Stretch* fields, type the stretch adjustments for X and Y.
8. Click *OK* and press `Escape` to end the command.

Repeat in X and Y Example



The shaded rectangle and two paths are repeated in X and Y with the following values:

X Stepping Distance = 3.0
Y Stepping Distance = 3.0
Number of X Repetitions = xRepeats
Number of Y Repetitions = yRepeats
Dependent X Stretch = stretch W
Dependent Y Stretch = stretch L
Adjustment to X Stretch = $\text{fix} (pcRepeatX-1) * pcStepX$
Adjustment to Y Stretch = $\text{fix} (pcRepeatY-1) * pcStepY$



Instance: xRepeats = 8; yRepeats = 2

Repeat in X and Y SKILL Function

```
pcHIDefineRepeat( "2D" ) => t | nil
```

Modify

Changes a previously defined repetition parameter; changes the objects in the repetition group or parameters assigned to the repetition group.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

To use Modify,

1. In a layout window, choose *Pcell – Repetition – Modify*.

The system prompts you to select an object in the repetition group whose definition you want to modify.

2. Click an object in the repetition group.

The system highlights the object. The system also highlights the rest of the objects in that repetition group.

The system prompts you to select objects you want to add to or remove from the repetition group.

3. Do one of the following:

- ☐ To remove an object from the repetition group, click the object so it is no longer selected.
- ☐ To include an object in the group, click it so that it is selected.

4. Double-click, or press `Return`, outside the objects to stop selecting.

The Modify Repeat in X form, Modify Repeat in Y form, or Modify Repeat in X and Y form appears.

The title of the form changes depending on what you select. These values were defined using the Repeat in X, Repeat in Y, or Repeat in X and Y form.

5. Do one of the following:

- ☐ If you want to keep the original values for the repetition, so that only the objects in the repetition group are modified, click *Cancel*.
- ☐ If you want to change the values, type in new values and click *OK* and press `Escape` to end the command.

If the object you selected in step 2 is associated with more than one repetition group, the system highlights the next repetition group and prompts you to point to objects you want to add to or remove from the group.

Modify Repeat SKILL Function

```
pcHIModifyRepeat( ) => t | nil
```

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

Delete

Deletes a repetition parameter assigned to a group of objects; deletes only the parameter assigned to the objects, not the objects themselves.

To use Delete,

1. In a layout window, choose *Pcell – Repetition – Delete*.

The system prompts you to select any member of the repetition group whose definition you want to delete.

2. Click an object in the repetition group.

The system highlights all objects in that repetition group and displays a Delete form showing information about the group.

The title of the form changes depending on what you select. These values were defined using the Repeat in X, Repeat in Y, or Repeat in X and Y command.

If there is more than one repetition group, the form displays information about only one of the groups.

3. Click the option you want for the group:

- ☐ *OK* closes the form and deletes the repetition group.
- ☐ *Cancel* closes the form and does not delete the repetition group.
- ☐ *Apply* deletes the repetition group and displays the information for the next repetition group.
- ☐ *Next* displays the information for the next repetition group and does not delete the first group.

Delete Repeat SKILL Function

```
pcHIDeleteRepeat( ) => t | nil
```

Show

Highlights objects in a repetition group and displays information about the group.

To use Show,

1. In a layout window, choose *Pcell – Repetition – Show*.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

The system highlights the objects in a repetition group and displays a window showing information about the group.

2. Do one of the following:

- ☐ If there is more than one repetition group, click *OK* to view the next group. When you are finished viewing, close the window by clicking *Cancel*.
- ☐ If there is only one repetition group, close the window by clicking *Cancel*.

Show Repeat SKILL Function

```
pcHIDisplayRepeat( ) => t | nil
```

Parameterized Shapes Commands

The Virtuoso® Parameterized Cell (Pcell) program lets you define Pcells containing shapes whose vertexes are parameters of the Pcell. When you place an instance of the Pcell, you supply values for the parameters by entering coordinates. Polygons, paths, and rectangles can be parameterized shapes.


- Creating Parameterized Shapes
- Defining a Margin
- Using the *Parameterized Shapes* menu, including
 - Creating a parameterized polygon, path, or rectangle or modifying the parameters assigned to an existing shape (see “Define/Modify” on page 104)
 - Removing the parameters from a shape (see “Delete” on page 106)
 - Displaying parameter information about a parameterized shape (see “Show” on page 107)

Virtuoso Parameterized Cell Reference

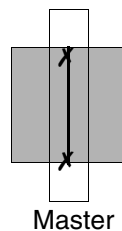
Creating Graphical Parameterized Cells

Creating Parameterized Shapes

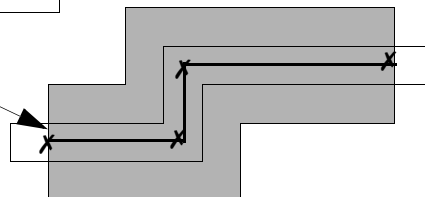
In the following example, poly and diffusion are defined as paths with different widths and coincident vertexes. Extensions are represented in the instance only if you define the appropriate path type in the Create Path form. You cannot represent extensions by drawing one path longer than another, because parameterized paths must have coincident vertexes. When you place the instance, you draw coordinates for the vertexes of the paths. The first coordinate you enter is the origin of the instance.

Diff 
End Type = variable
Begin and End Extensions = 0
Width = 6


Create Path			
Hide	Cancel	Defaults	Help
Mode	<input checked="" type="radio"/> Guided	<input type="radio"/> Manual	Change To Layer
Width	<input type="text" value="6"/>	<input type="text" value="None"/>	
Fixed Width	<input type="checkbox"/>	Contact Justification	
Offset	<input type="text" value="0"/>	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	
Justification	<input type="text" value="center"/>	<input type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/>	
End Type	<input type="text" value="variable"/>	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	
Begin Extension	<input type="text" value="0"/>	Snap Mode <input type="text" value="orthogonal"/>	
End Extension	<input type="text" value="0"/>		



Origin of cellview



Instance

Poly 
End Type = variable
Begin and End Extensions = 2
Width = 2

Create Path			
Hide	Cancel	Defaults	Help
Mode	<input checked="" type="radio"/> Guided	<input type="radio"/> Manual	Change To Layer
Width	<input type="text" value="2"/>	<input type="text" value="None"/>	
Fixed Width	<input type="checkbox"/>	Contact Justification	
Offset	<input type="text" value="0"/>	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	
Justification	<input type="text" value="center"/>	<input type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/>	
End Type	<input type="text" value="variable"/>	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	
Begin Extension	<input type="text" value="2"/>	Snap Mode <input type="text" value="orthogonal"/>	
End Extension	<input type="text" value="2"/>		

Defining a Margin

You can offset a parameterized shape from the points you enter to draw it by associating a *margin* with the shape when you define the Pcell. Margins are supported only when the shapes are drawn with orthogonal vertexes.

Using a Margin with Parameterized Polygons

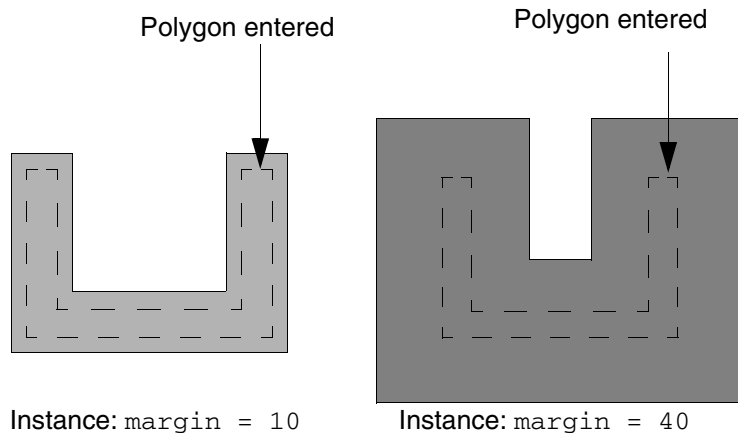
A margin in parameterized polygons and rectangles makes the shape in the instance bigger or smaller than the shape you draw. The size and shape of the object in the master is irrelevant in parameterized polygons because the shape is determined when the instance is placed, and the size is generated by adding the margin to the dimensions.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

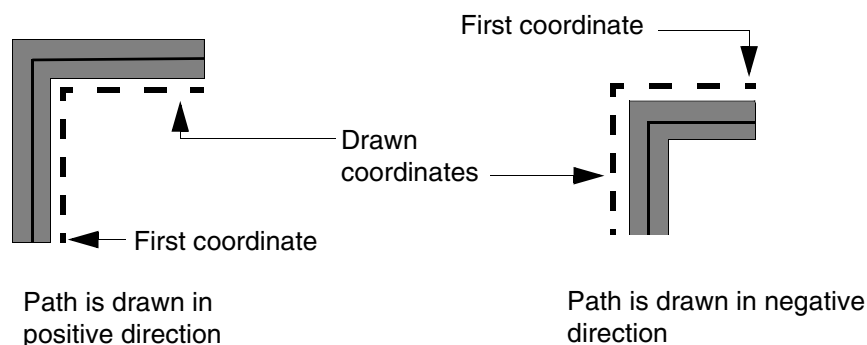
The margin is optional for parameterized polygons. If you use the default margin (0) when you create the Pcell, the parameterized polygon is created as it is drawn when the instance is placed.

The figure shows how this works for two instances of a parameterized polygon. Both polygons are entered with the same coordinates; only the margins are different.



Using a Margin with Parameterized Paths

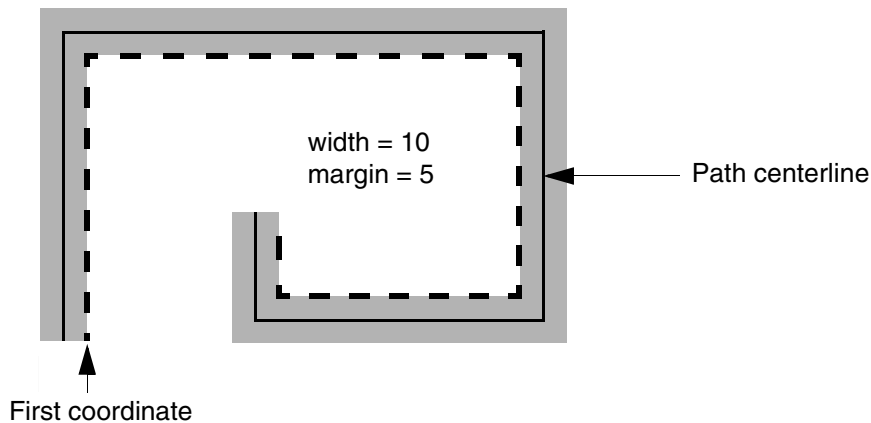
When you use a margin with a parameterized path, the effect of the margin is to offset the centerline of the path to one side or another of the coordinates you enter. The value of the margin is the distance between the coordinates you draw and the centerline of the path. The effect of the margin depends on the direction of the path segment and the sign (+ or -) and magnitude of the margin. For a positive margin, the path is generated above and to the left of a segment drawn in a positive direction (left to right or bottom to top). The path is generated below and to the right of a segment drawn in a negative direction (top to bottom or right to left).



Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

If the width of the path is twice the value of the margin, the edge of the generated path falls on the path you draw. That is, you draw the edge of the path.



If you have more than one parameterized path in the Pcell, you can still have only one width parameter if the widths of the other paths in the Pcell are defined as functions of the single width parameter. For example, the parameter name for the width of poly of a transistor can be `width`, and the expression for the width of the diffusion of the transistor can be `width+4`. In this case, when you place the Pcell, you are prompted for only the width of the poly path.

Parameterized Shapes Menu

Parameterized Shapes commands set a parameter to create customized polygons, paths, and rectangles that you draw when you place an instance. Use *Define/Modify* to create a parameterized polygon, path, or rectangle or to modify the parameters assigned to an existing shape. Use *Delete* to remove the parameterization from a shape. Use *Show* to display information about a parameterized shape.

Define/Modify

Assigns the vertexes of a shape as parameters of the Pcell. You can define paths, polygons, and rectangles as parameterized shapes.

To use *Define/Modify*,

1. In a layout window, choose *Pcell – Parameterized Shapes – Define/Modify*.

The system prompts you to select a shape to define or modify.

2. Click the shape whose vertexes you want to parameterize.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

The system highlights the shape, and the Define Parameterized Path form, Define Parameterized Polygon form, or Define Parameterized Rectangle form appears.

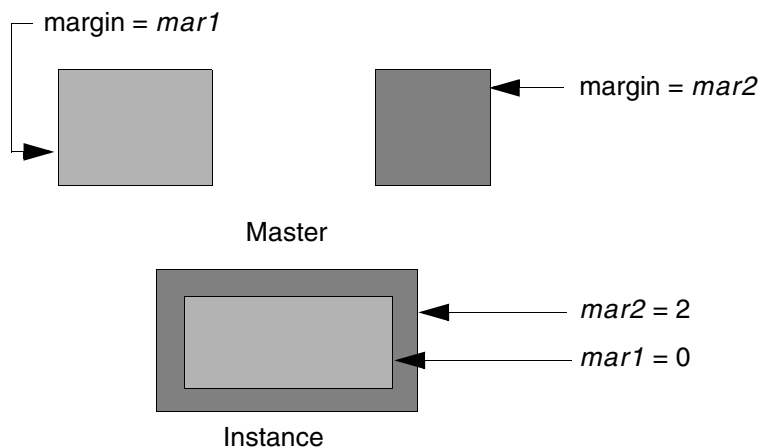
3. Change values in the appropriate fields.

Margins are optional. If you are defining a parameterized path, you must type a value in the *Width* field.

4. Click *OK* and press `Escape` to end this command.

Define Parameterized Shape Examples

Multiple polygons do not have to be drawn in any relative location in the master. When you specify coordinates for the Pcell instance, all parameterized polygons use these coordinates by default. The polygons are then sized according to their respective margin values.



Multiple rectangles follow the same rules as multiple polygons.


Multiple parameterized paths in a Pcell must have the same vertexes. Extensions are created by using the appropriate path type for each path entered.


Parameterized path masters can be tricky to create. Paths must all have coincident vertexes. If you want one path to extend past the end of another, as the poly of a transistor extends past the diffusion, you cannot draw it longer in the master. When you draw your paths, use the appropriate path type to establish the extension and enter the appropriate values for *Begin Extension* and *End Extension* fields in the Create Path form.

Virtuoso Parameterized Cell Reference

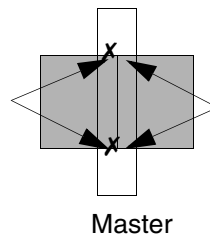
Creating Graphical Parameterized Cells

A parameterized path with a margin value can be useful when the minimum path width forces the path centerline off grid. A margin of half the path width lets you draw the edge of the path instead of the centerline when you enter coordinates.

Diff 
 End Type = variable
 Begin and End Extensions = 0
 Width = 8

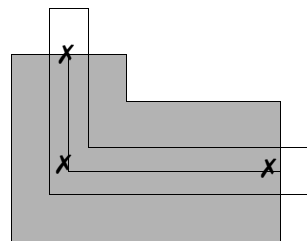
Poly 
 End Type = variable
 Begin and End Extensions = 2
 Width = 2

Create Path			
Hide Cancel Defaults		Help	
Mode	<input checked="" type="radio"/> Guided <input type="radio"/> Manual	Change To Layer	
Width	<input type="text" value="8"/>	<input type="text" value="None"/>	
Fixed Width	<input type="checkbox"/>		
Offset	<input type="text" value="0"/>	Contact Justification	
Justification	<input type="text" value="center"/>	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	
End Type	<input type="text" value="variable"/>		
Begin Extension	<input type="text" value="0"/>		
End Extension	<input type="text" value="0"/>	Snap Mode <input type="text" value="orthogonal"/>	



Master

Create Path			
Hide Cancel Defaults		Help	
Mode	<input checked="" type="radio"/> Guided <input type="radio"/> Manual	Change To Layer	
Width	<input type="text" value="2"/>	<input type="text" value="None"/>	
Fixed Width	<input type="checkbox"/>		
Offset	<input type="text" value="0"/>	Contact Justification	
Justification	<input type="text" value="center"/>	<input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	
End Type	<input type="text" value="variable"/>		
Begin Extension	<input type="text" value="2"/>		
End Extension	<input type="text" value="2"/>	Snap Mode <input type="text" value="orthogonal"/>	



Instance

Define Parameterized Shape SKILL Function

```
pcHIDefineParameterizedShape( ) => t | nil
```

Delete

Deletes the parameters associated with a polygon, path, or rectangle.

To use Delete,

1. In a layout window, choose *Pcell – Parameterized Shapes – Delete*.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

The system prompts you to select the shape whose parameters you want to delete.

2. Click the shape whose coordinate parameters you want to delete.

The system displays a Delete Parameterized Path form, Delete Parameterized Polygon form, or a Delete Parameterized Rectangle form, showing information about the parameterized shape.

3. Do one of the following:

- ☐ If you do not want to delete the coordinate parameters for the shape, click *Cancel*.
- ☐ If you want to delete the coordinate parameters for the shape, click *OK*.

Parameterized Shape Delete SKILL Function

```
pcHIDeleteParameterizedShape( ) => t | nil
```

Show

Highlights a parameterized shape and displays information about the shape and its parameters.

To use Show,

1. In a layout window, choose *Pcell – Parameterized Shapes – Show*.

The system highlights a parameterized shape and displays a window showing information about the shape.

2. Do one of the following:

- ☐ If there is more than one shape, click *OK* to view the next shape. When you are finished viewing, close the window by clicking *Cancel*.
- ☐ If there is only one shape, close the window by clicking *Cancel*.

Show Parameterized Shape SKILL Function

```
pcHIDisplayParameterizedShape( ) => t | nil
```

Repetition Along Shape Commands

The Virtuoso® Parameterized Cell (Pcell) program lets you repeat figures along the length of a parameterized shape. This gives the effect of repetition along the length of a parameterized path or around the perimeter of a parameterized polygon or rectangle.

- Using Control Path Segments
- Specifying Start and End Offsets
- Using the *Repetition Along Shape* menu, including
 - ❑ Assigning an object to be repeated along a parameterized shape (see “Define” on page 112)
 - ❑ Changing the objects in a repetition group or the values that control the repetition, such as stepping distance or start and end offsets (see “Modify” on page 114)
 - ❑ Removing the repetition along shape parameters from an object (see “Delete” on page 115)
 - ❑ Displaying parameter information about an object to be repeated along a shape (see “Show” on page 115)

Using Control Path Segments

The rules that define the relationship between the objects and the parameterized shape in the master cellview are as follows:

- The Pcell program considers only one segment of the shape to be the control path segment.
- If the control shape is a polygon or rectangle, the lower leftmost segment is the control path segment.
- The control path segment is either the first or last segment of a parameterized path, depending on which endpoint of the path you reference.
- The bottom vertex of the control path is the first vertex and the top vertex is the last.
- Only horizontal or vertical segments are used in the control path segment.
- Usually, the parameterized shape in the master cellview has only a single segment.

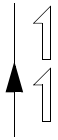
When you rotate the control path segment counterclockwise about the origin, the objects defined with reference to the control path endpoint rotate the same way. The Pcell program maintains the relationship between the rotated object and the rotated control path segment.

Virtuoso Parameterized Cell Reference

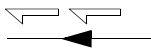
Creating Graphical Parameterized Cells

The Pcell program orients the repeated objects in the placed instance as follows:

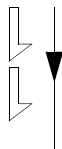
- For a segment drawn bottom to top



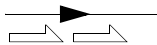
- For a segment drawn right to left



- For a segment drawn top to bottom



- For a segment drawn left to right



The coordinates of the control path segment and the repeated objects in the master cellview are not relevant; only their relative locations are important.

Specifying Start and End Offsets

When you define a repetition along a parameterized shape, you specify the following:

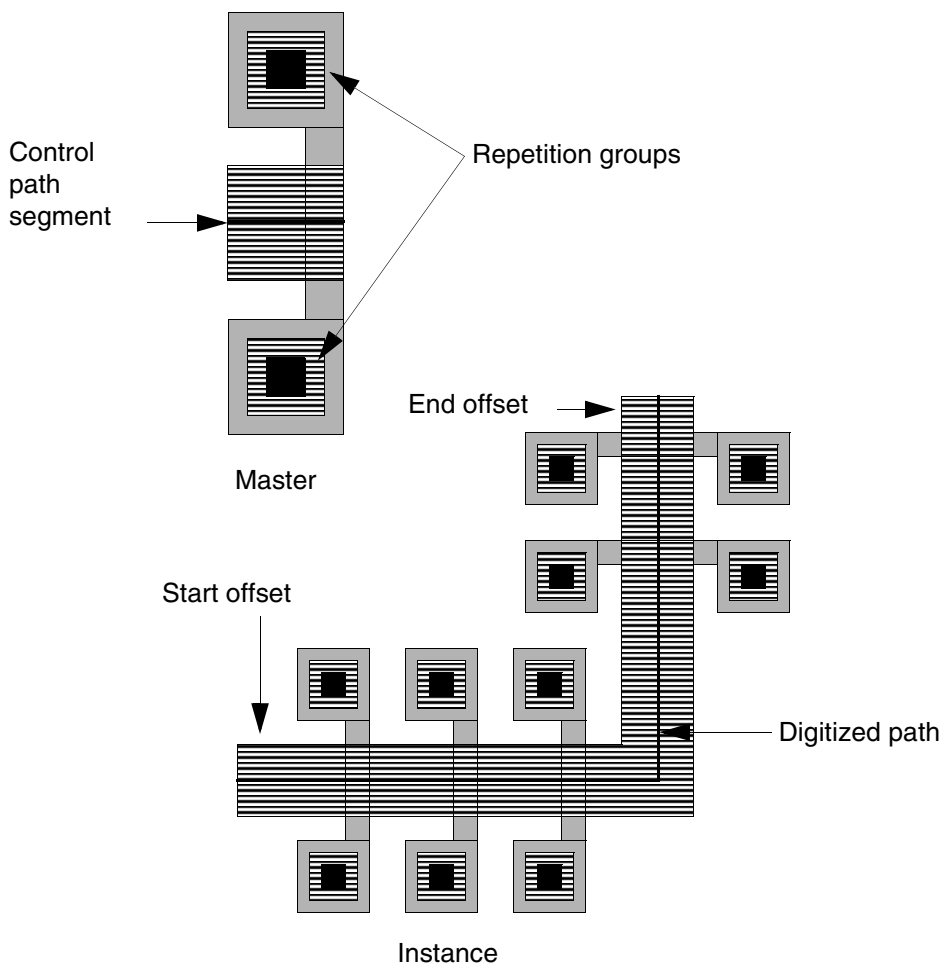
- Stepping distance between the repeated figures
- Options for start and end offsets

When you supply values for *Start Offset* and *End Offset* in the Repetition Along Shape form, the Pcell program repeats the figures along the length of the digitized shape, leaving a gap between the first point entered and the first repeated figure and between the last point entered and the last repeated figure. If you want the repetition of the figures to start before the first point drawn or end after the last point drawn, use negative values for *Start Offset* and *End Offset*.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

The relative position of the repeated objects to the parameterized shape remains unchanged when you place an instance. All objects in a repetition group must use the same start and end offsets. To repeat the objects shown in the example, you need to define four separate repetition groups: one group for the contacts, one for the metal around the contacts, one for the poly around the contacts, and one for the poly rectangle that crosses the metal path.

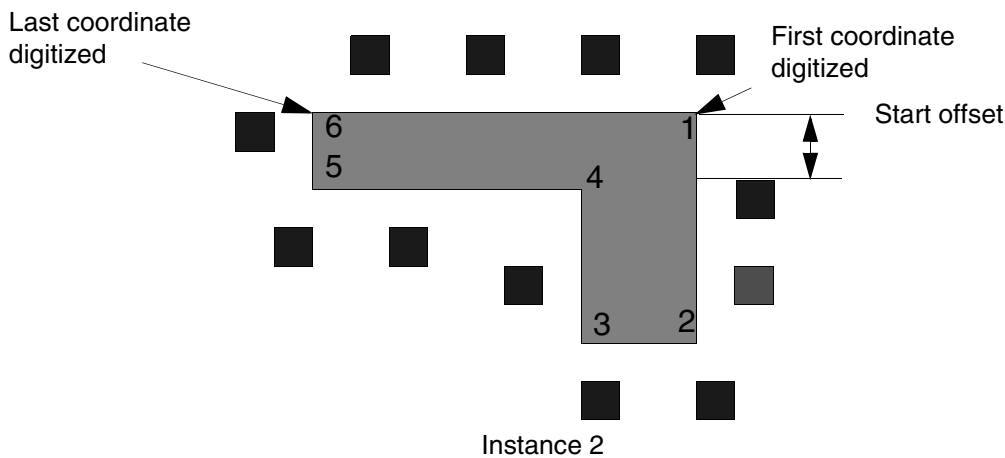
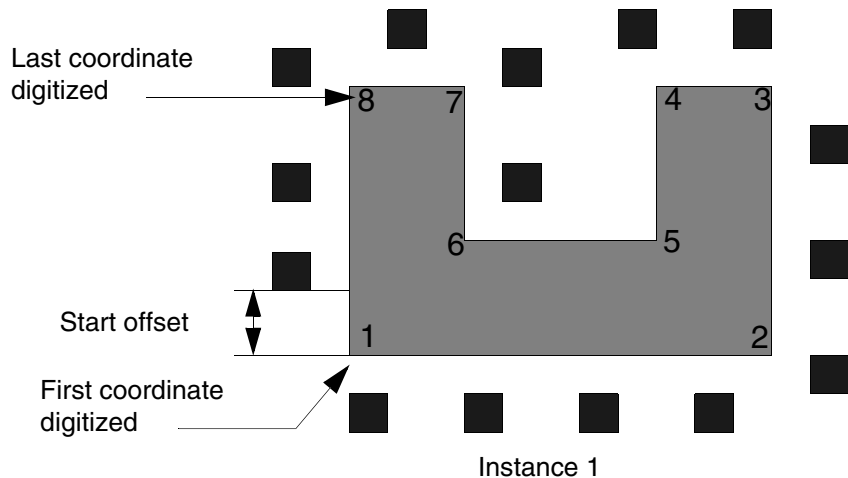


When you place an instance of a parameterized polygon, you define a coordinate path that begins and ends at the first vertex of the polygon. You enter these coordinates by drawing them or typing them in the *coords* field on the Create Instance form. Regardless of whether you enter the coordinates in a clockwise or counterclockwise direction, the program always repeats objects around a parameterized polygon in a clockwise direction. This ensures that a shape drawn inside the polygon in the master cellview is repeated around the inside of the polygon in every instance.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

No matter what order you use to draw the vertexes of the polygon, *Start Offset* is measured in a clockwise direction from the first vertex digitized.

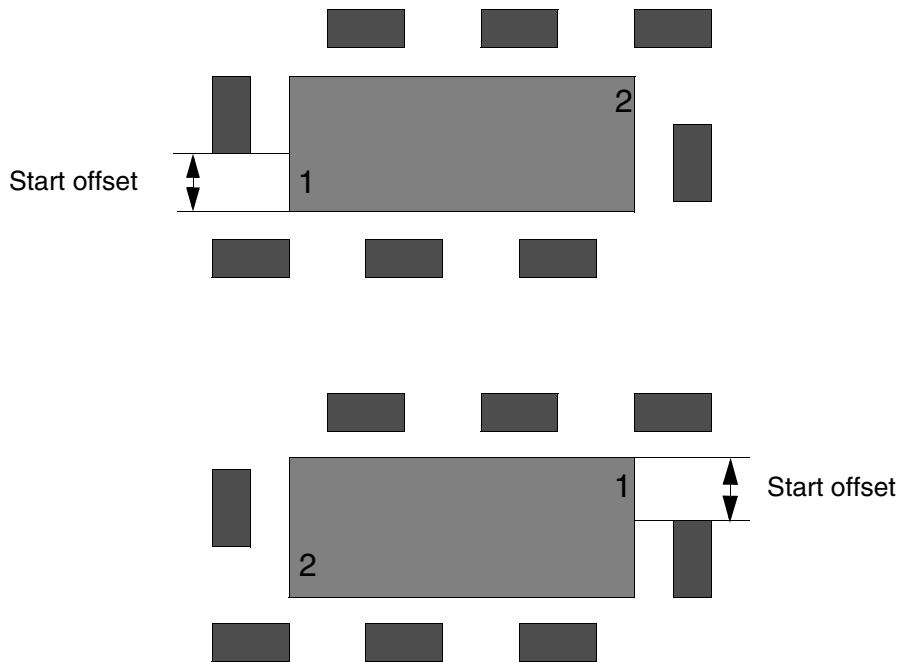


When you place an instance of a parameterized rectangle, you enter the two opposite vertexes of the rectangle. The first vertex of the rectangle defines the startpoint and endpoint for repetition of objects around the rectangle. As with polygons, repetition around a rectangle

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

always takes place in a clockwise direction. *Start Offset* is always calculated in a clockwise direction from the first point digitized.



Repetition Along Shape Menu

Repetition Along Shape commands create a parameter that repeats objects along a polygon, path, or rectangle. Any object can be repeated along the interior or exterior of any parameterized shape. Use *Define* to assign an object to be repeated along a parameterized shape. Use *Modify* to change the objects in a repetition group or the values that control the repetition, such as stepping distance or start and end offsets. Use *Delete* to remove parameters from a repetition along shape group. Use *Show* to display information about a repetition along shape group.

Define

Repeats an object or group of objects along a coordinate string controlling a parameterized shape.

Note: Before you can repeat objects along a shape, you must create and define a parameterized shape with *orthogonal* snap.

To use Define,

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

You can preselect objects or select them after starting the command.

1. In a layout window, choose *Pcell – Repetition Along Shape – Define*.

If you have not preselected any objects, the system prompts you to select the objects to repeat along the parameterized shape.

If you have not defined a parameterized shape, or if any parameterized shape you select does not have an orthogonal snap, the system issues a warning message in the CIW and you cannot complete the command.

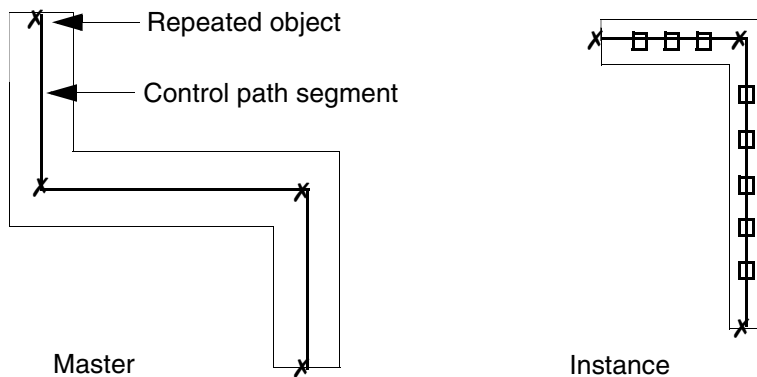
2. Click the objects you want to repeat.
3. Double-click, or press `Return`, to stop selecting.

The Repetition Along Shape form appears.

If you select an object that is already associated with a parameterized reference point or already defined as repeated along a parameterized shape, the program warns you and does not add the object to the repetition group.

4. In the *Stepping Distance*, *Start Offset*, and *End Offset* fields, type values. You can accept the default (0) for *Start Offset* and *End Offset*.
5. Click *OK* and press `Escape` to end the command.

Repetition Along Shape Example



The path is parameterized and has the parameter name `coords`. The rectangle is repeated along a parameterized shape with the following values:

Parameter Name = `coords`
Stepping Distance = 3

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

Start Offset = 1
End Offset = 1

Repetition Along Shape Define SKILL Function

```
pcHIDefineSteppedObject( ) => t | nil
```

Modify

Changes a previously defined *Repetition Along Shape* parameter; changes the objects to be repeated along the parameterized shape or the parameterized shape itself.

To use Modify,

1. In a layout window, choose *Pcell – Repetition Along Shape – Modify*.

The system prompts you to select an object in the repetition along shape group whose definition you want to modify.

2. Click an object in the repetition along shape group.

The system highlights all the objects in that repetition group.

The system prompts you to select objects you want to add to or remove from the repetition group.

3. Do one of the following:

- ☐ To remove an object from the repetition along shape group, click the object so it is no longer selected.
- ☐ To include an object in the group, click it so that it is selected.

4. Double-click, or press `Return`, to stop selecting.

The Modify Repetition Along Shape form appears.

5. Do one of the following:

- ☐ If you want to keep the original values and change only the objects in the repetition group, click *Cancel*.
- ☐ Type in new values and click *OK*.

Modify Repetition Along Shape SKILL Function

```
pcHIModifySteppedObject( ) => t | nil
```

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

Delete

Deletes a *Repetition Along Shape* parameter; deletes only the parameters assigned to the shape, not the shape itself.

To use Delete,

1. In a layout window, choose *Pcell – Repetition Along Shape – Delete*.

The system prompts you to select an object in the repetition along shape group whose definition you want to delete.

2. Click an object in the group.

The program highlights the object. The system also highlights the rest of the objects in that repetition along shape group and displays the Delete form showing information about the group.

3. Do one of the following:

- ☐ If you do not want to delete the repetition along shape definition, click *Cancel*.
- ☐ If you want to delete the repetition along shape definition, click *OK*.

Delete Repetition Along Shape SKILL Function

```
pcHIDeleteSteppedObject( ) => t | nil
```

Show

Highlights objects in a *Repetition Along Shape* group and displays information about the group.

To use Show,

1. In a layout window, choose *Pcell – Repetition Along Shape – Show*.

The system highlights the objects in a repetition along shape group and displays the Show Repetition Along Shape window with information about the group.

2. Do one of the following:

- ☐ If there is more than one repetition along shape group, click *OK* to view the next group. When you are finished viewing, close the window by clicking *Cancel*.
- ☐ If there is only one repetition along shape group, close the window by clicking *OK*.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

Show Repetition Along Shape SKILL Function

```
pcHIDisplaySteppedObject( ) => t | nil
```

Reference Point Commands

The Virtuoso® Parameterized Cell (Pcell) *Reference Point* commands let you select objects whose location in a placed instance remains relative to a specified reference point in the master cellview. Using the commands on this menu, select the objects and specify which end of the path or which parameter the objects are to remain relative to. When you place the instance, the objects you select keep the same relationship to the reference point as in the master cellview. You can define only one reference point by parameter and only one reference point by path endpoint for a Pcell.

- Using a Reference Point Defined by Path Endpoint
- Using a Reference Point Defined by Parameter
- Using the *Reference Point* menu, including
 - Creating a group of objects whose relative location to a parameterized path endpoint remains constant in any instance (see “Define by Path Endpoint” on page 121)
 - Creating a group of objects whose location in an instance is controlled by a reference point (see “Define by Parameter” on page 123)
 - Changing the group of objects or the reference parameter (see “Modify” on page 124)
 - Removing the reference parameter from a group of objects (see “Delete” on page 125)
 - Displaying reference parameters for a group of objects (see “Show” on page 126)

Using a Reference Point Defined by Path Endpoint

You can specify a reference point so that when you place an instance, the location of the objects relative to a path endpoint in the placed instance remains the same as the location in the master cellview.

The rules that define the relationship between the objects and the parameterized path in the master cellview are as follows:

- The Pcell program considers only one segment of the path to be the control path.
- The control path is either the first or last segment of the parameterized path, depending on which endpoint of the path you specify.
- Use only horizontal or vertical segments in the control path. Usually, the parameterized path in the master cellview has only a single segment.

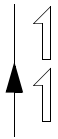
Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

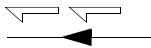
When you rotate the control path segment counterclockwise about the origin until the control path segment is vertical, the objects defined with reference to the path endpoint rotate the same way. The Pcell program maintains the relationship between the rotated object and the rotated control path segment.

The Pcell program orients repeated objects in the placed instance as follows:

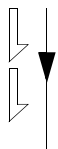
- For a segment drawn bottom to top



- For a segment drawn right to left



- For a segment drawn top to bottom



- For a segment drawn left to right



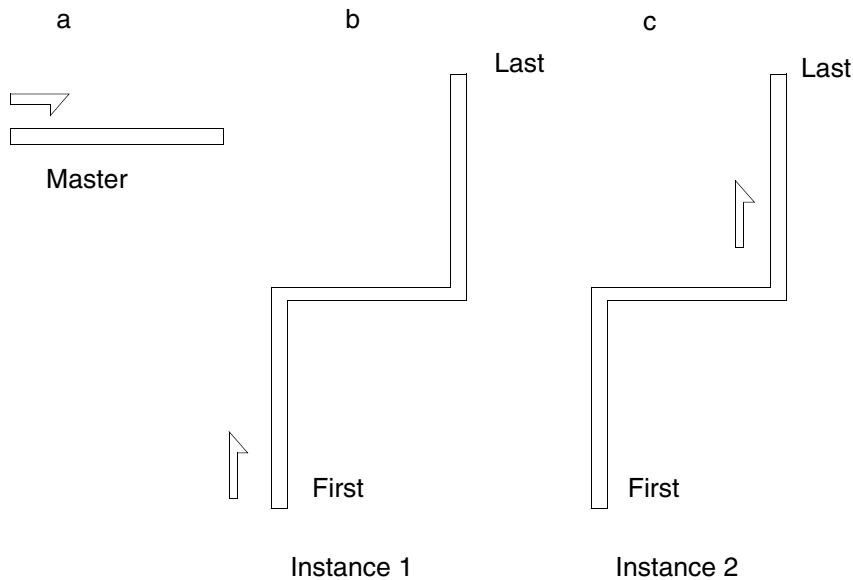
In the following figure,

- (a) shows the relationship in the master cellview between a parameterized path and an object to be placed relative to an endpoint of the path
- (b) shows the location of the object in an instance of the cellview when the object is defined relative to the first endpoint of the parameterized path

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

- (c) shows the location of the object in an instance with the same parameterized path coordinates when the object is defined relative to the last endpoint of the path



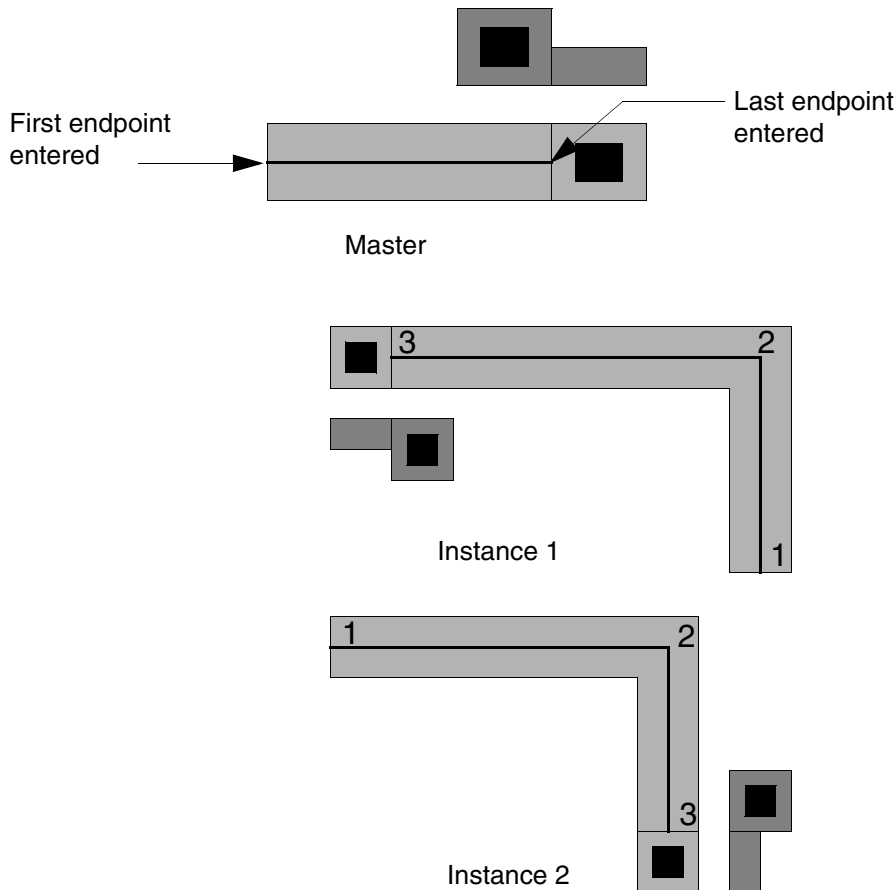
Only the relative locations of the control path segment and the repeated objects in the master cellview are relevant, not the coordinates entered. The order in which you enter the coordinates of the control path is only relevant in determining which is the first or last segment. You choose the first or last segment as the control path segment when you define the reference point parameter.

The following examples show two contacts defined by path endpoint in the master cellview relative to the last endpoint of a parameterized path. In the first instance, the parameterized path is digitized bottom to top. In the second instance, the parameterized path is digitized left

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

to right. In each instance, the two contacts keep the same relationship to the last endpoint as in the master cellview.



Using a Reference Point Defined by Parameter

You can specify the location of the reference point as a parameter of the cell. This parameter is a standalone coordinate not tied to any graphic object in the cell.

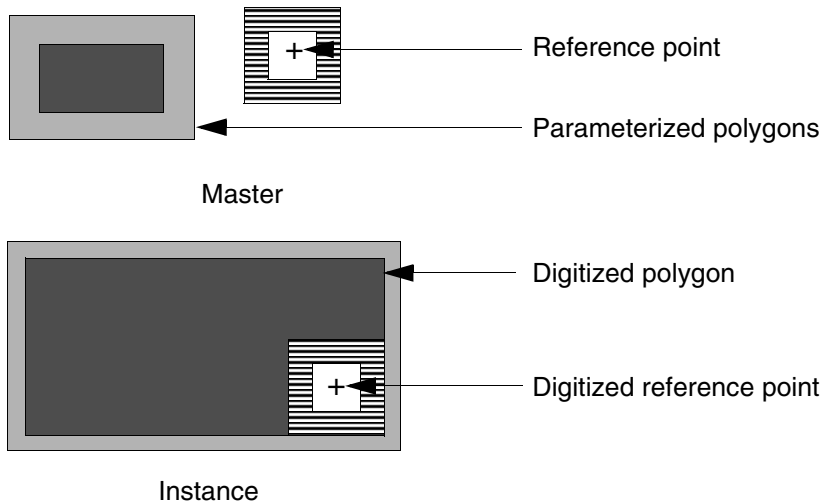
The system prompts you to point to a location in the master cellview to define as the master origin and another location that you define as the parameterized origin (reference point). When you place an instance of the cellview, you are prompted to enter a coordinate for the origin of the instance and the vertexes of the parameterized shape.

After you draw a parameterized path, polygon, or rectangle, you are prompted to enter a coordinate for the parameterized origin. All objects associated with this parameterized origin are placed relative to the coordinate you enter, regardless of the location of other objects in the instance or the origin of the instance.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

When placing this type of Pcell, you enter multiple origins: one for all objects not associated with a reference point parameter and another for all objects that are associated with the reference point parameter.



Reference Point Menu

Reference Point commands create a parameter that specifies the location of objects relative to a parameterized reference point or the endpoint of a parameterized path. Use *Define by Path Endpoint* to place objects relative to the endpoint of a parameterized path. Use *Define by Parameter* to place objects relative to a parameterized reference point. Use *Modify* to change either type of reference point. Use *Delete* to remove either type of reference point. Use *Show* to display information about the reference points.

Define by Path Endpoint

Specifies that the location of an object or group of objects is determined by the location of the endpoint of a parameterized path. You can define only one reference point by path endpoint for a Pcell.

Note: Before you define a reference point, you must create a path and define it as a parameterized path.

To use Define by Path Endpoint,

You can preselect the objects or select them after starting the command.

1. In a layout window, choose *Pcell – Reference Point – Define by Path Endpoint*.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

If you have not preselected any objects, the system prompts you to select the objects to include in the reference point group.

2. Click the objects you want to include.

These are the objects that are placed relative to an endpoint of the parameterized path in the instance.

3. Double-click, or press `Return`, to stop selecting.

The Reference Point by Path Endpoint form appears.

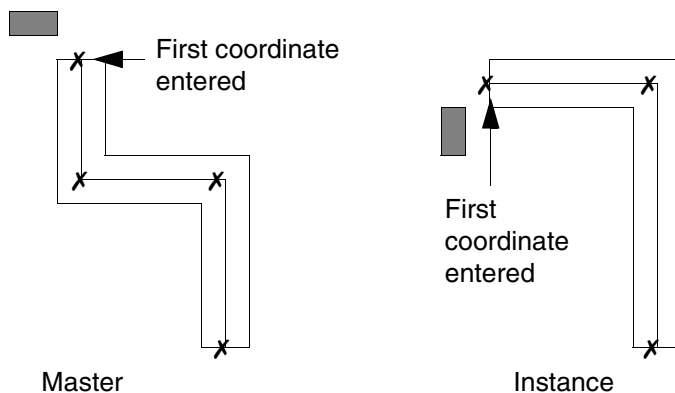
4. Specify the endpoint of the path by clicking *first* or *last*.
5. Click *OK* and press `Escape` to end the command.

Defining Paths

You must use only horizontal or vertical segments in the control path. Usually, the path has only a single segment because more segments are irrelevant.

When you place an instance of the Pcell, the system prompts you to enter the points of the parameterized path. The objects are placed in the same relative location to the path as in the master cellview. The system looks at the orientation of the first or last segment of the path to determine the relative location of the objects.

Define Reference Point by Path Example



The path is parameterized with the parameter `coords`. The shaded rectangle is defined in reference to a path endpoint group with the following values:

Path Parameter Name = `coords`
Endpoint of the Path = `first`

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

Define Reference Point by Path SKILL Function

```
pcHIDefinePathRefPointObject( ) => t | nil
```

Define by Parameter

Specifies that the location of an object or group of objects is determined by a reference point parameter in the cell. In addition to the master origin of the cell, you can specify a parameterized origin for the associated objects. You can only define reference point by parameter for a Pcell.

To use Define by Parameter,

You can preselect the objects or select them after starting the command.

1. In a layout window, choose *Pcell – Reference Point – Define by Parameter*.

If you have not preselected any objects, the system prompts you to select the objects to include in the reference point group.

2. Click each object you want to include.
3. Double-click, or press `Return`, to stop selecting.

If you click an object that is already associated with a parameterized reference point, the system issues a warning message in the CIW and does not select the object.

The system prompts you to select a location to be used as the reference point for the objects.

4. Click the location for the reference point.

The system draws a square around the location.

The Reference Point by Parameter form appears.

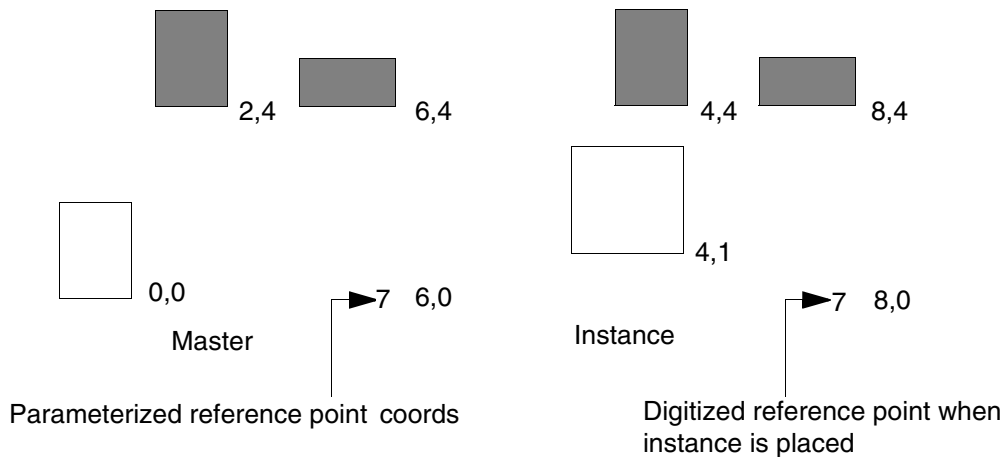
5. In the *Parameter Name* field, type a parameter name.
6. Click *OK* to close the form.

This command is not repetitive because you can have only one reference point defined by parameter in a Pcell.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

Define by Parameter Example



The shaded rectangles are defined with respect to the reference point parameter.

In this example, the Pcell containing the reference point also contains a parameterized rectangle (unshaded). When you place an instance of this Pcell, the system prompts you to enter coordinates for the parameterized rectangle. After you draw the rectangle, the system prompts you for the location of the reference point. The objects associated with the reference point (shaded) are placed relative to the reference point. The location of the parameterized rectangle has no relation to the locations of the reference point or the rectangles associated with the reference point.

Define by Parameter SKILL Function

```
pcHIDefineParamRefPointObject( ) => t | nil
```

Modify

Modifies a reference point parameter. You can use this command to modify either a reference point defined by the [Reference Point by Parameter form](#) or the [Reference Point by Path Endpoint form](#).

To use Modify,

1. In a layout window, choose *Pcell – Reference Point – Modify*.

The system prompts you to select an object in the reference point group that you want to modify.

2. Click an object in the reference point group.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

The system highlights all objects in that reference point group. The system then prompts you to point to objects in the reference group that you want to include in or exclude from the reference group.

3. Click the objects you want to select.

Each object is selected as you click it. Once an object is selected, you can deselect it by pressing `Control` and clicking on it again.

4. Double-click, or press `Return`, to stop selecting objects.

5. Do one of the following:

- ☐ If this reference point group is associated with a reference point parameter, the program highlights the parameterized reference point. The system prompts you to point to a new parameterized reference point for the selected objects.

a. Click the location for the parameterized reference point.

The Reference Point by Parameter form appears.

b. Type or change the values for the parameter.

c. Click `OK` to close the form and end this command.

- ☐ If the reference point group is associated with a parameterized path, the program displays the Reference Point by Path Endpoint form.

a. Select either the first or last endpoint of the path as the reference parameter. You cannot change the path parameter name.

b. Click `OK` to close the form and end this command.

Note: This command is not repetitive because you can only have one reference point parameter of each type in a Pcell.

Modify Reference Point SKILL Function

```
pcHIModifyRefPointObject( ) => t | nil
```

Delete

Deletes a previously defined reference point parameter. Use this command to delete either a reference point defined as a parameter of the cellview or a reference point defined relative to a parameterized path endpoint.

To use Delete,

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

1. In a layout window, choose *Pcell – Reference Point – Delete*.

The system prompts you to select any member of the reference point group whose reference point you want to delete.

2. Click an object in the reference point group.

The system highlights all objects in that reference point group and either the Delete Reference Point by Path form or the Delete Reference Point form appears showing information about the group.

If the object you select is not associated with a reference point parameter, the system issues a warning message in the CIW and cannot complete the command.

3. Do one of the following:

- ☐ If you do not want to delete the reference point parameter for the group of objects, click *Cancel*.
- ☐ If you want to delete the reference point parameter for the group of objects, click *OK*.

Delete Reference Point SKILL Function

```
pcHIDeleteRefPointObject( ) => t | nil
```

Show

Highlights objects in a reference point group and displays information about the group.

To use Show,

1. In a layout window, choose *Pcell – Reference Point – Show*.

The system highlights the objects in a reference point group. If the reference point is a parameter of the cell, the system draws a small square around the parameterized reference point.

The Show Reference Point window appears, showing information about the reference point group.

2. Do one of the following:

- ☐ If there is more than one reference point group, click *OK* to view the next group. When you are finished viewing, close the window by clicking *Cancel*.
- ☐ If there is only one reference point group, close the window by clicking *OK*.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

Show Reference Point SKILL Function

```
pcHIDisplayRefPointObject( ) => t | nil
```

Inherited Parameters Commands

Inherited parameters let lower-level (child) cells *inherit*, or use, the parameters assigned to the Pcell (parent) in which they are placed. Using an inherited parameter, you control the parameter of the child cell when you place an instance of the parent cell. This allows you to use nested Pcells and maintain complete control over all the parameters without flattening data.

- Creating a Pcell with Inherited Parameters
- Using the *Inherited Parameters* menu, including
 - Choosing which parameters of the lower-level cell to inherit and giving the parameters new definitions (see “Define/Modify” on page 130)
 - Displaying information about a lower-level cell and its inherited parameters (see “Show” on page 131)

Creating a Pcell with Inherited Parameters

You might build a parent inverter using instances of child Pgate and Ngate Pcells. When placing an instance of the inverter cell, you can specify the gate width of the component P gates and N gates.

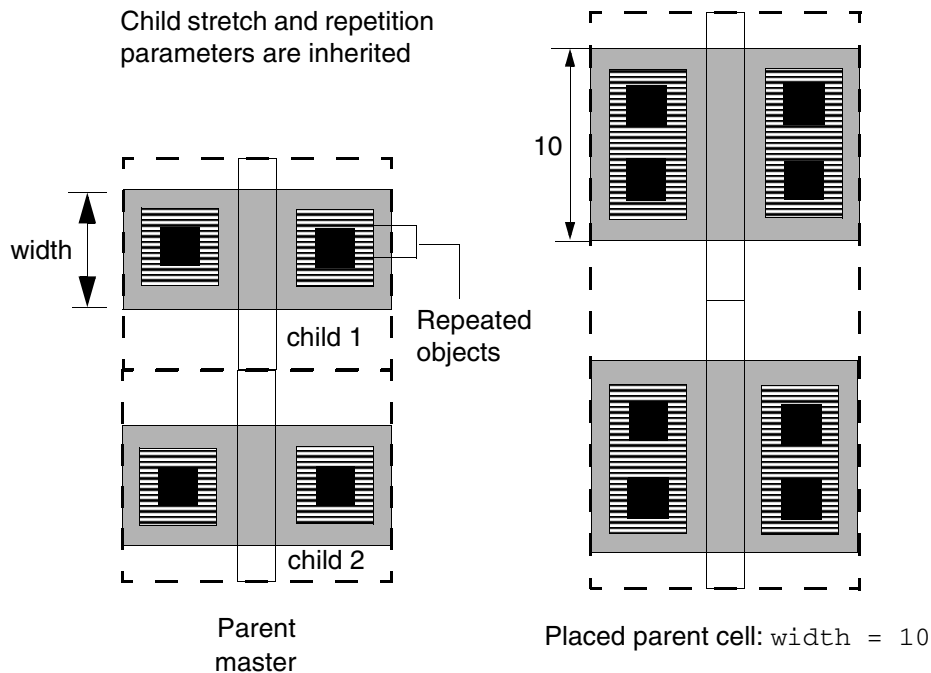
To control a parameter of a lower-level Pcell,

1. Place an instance of the lower-level Pcell in your parent design.
2. Select the lower-level Pcell and use the *Define/Modify Inherited Parameters* form to indicate which of its parameters you want to control from the parent Pcell.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

3. Specify the controlling expression. This expression becomes a parameter of the parent Pcell, and the value of the expression is passed down to the lower level Pcell when an instance of the parent Pcell is placed.



When the definition is a name, such as `invWidth`, you are prompted to give it a value when you place the parent Pcell. The definition can, however, be more complex than a single name. The definition can be any valid Cadence® SKILL language expression referencing one or more names, each of which becomes a parameter of the parent Pcell, such as

```
cellHeight - 2 * busWidth
```

When you place an instance of the parent Pcell, you are prompted to enter values for `cellHeight` and `busWidth`. The system uses the values you enter to evaluate the expression and then uses the computed value for the inherited parameter.

You can specify an expression that accesses values in the technology file without creating any new parameters in the parent Pcell, such as

```
techGetSpacingRule(  
    techGetTechFile(pcCellView)  
    "minWidth" list("poly" "drawing")  
)
```

This expression tells the system to find the minimum poly width for the current technology and use that value for the parameter in the child cell. When you place an instance of the parent Pcell, you are not prompted to enter any value for the parameter.

Inherited Parameters Menu

Inherited Parameters commands create a parameter that passes parent parameters to a child cell. Use *Define/Modify* to specify parameter values for lower-level Pcells when an instance of the higher-level Pcell is placed. Use *Show* to highlight and display information about lower-level Pcells.

Define/Modify

Specifies that the Pcell child gets some or all of its parameter values from the Pcell parent in which the child instance is placed. You can specify that individual parameter values of the parent Pcell be passed to the child Pcell when you place the Pcell parent in your design.

Note: Before you can define inherited parameters, you must place an instance of a child Pcell in the parent Pcell from which the child Pcell is to inherit the parameters.

To use Define/Modify,

1. In a layout window, choose *Pcell – Inherited Parameters – Define/Modify*.

The system prompts you to select a child Pcell instance whose parameters you want to be inherited from the parent Pcell.

2. Click the child Pcell instance.

The system highlights the selected instance.

The Define/Modify Inherited Parameters form appears, showing the parameter values for the child Pcell.

3. Do one of the following:

- ☐ To specify that a parameter value be inherited, click *inherit* next to the parameter and type a name or a SKILL expression for the value of the parameter.
- ☐ If you want to remove an inherited parameter that you have already defined, click *inherit* next to the parameter so it is no longer selected.
- ☐ If you want to modify an inherited parameter, click *inherit* next to the parameter and change the value of the parameter.

When you place an instance of the parent, the values you enter here are the parameter names that are displayed in the Create Instance form.

4. Click *OK* and then press *Escape* to end the command.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

Define Inherited Parameters SKILL Function

```
pcHIDefineInheritedParameter( ) => t | nil
```

Show

Highlights the child cell instances in an inherited parameter group and displays information about the group.

To use Show,

1. In a layout window, choose *Pcell – Inherited Parameters – Show*.

The system highlights the instances in the inherited parameter group and displays the Show Inherited Parameters window, showing information about the group. Notice that the name of the instance is not the name of the child master but the name assigned to the instance when it was placed in the parent Pcell.

2. Do one of the following:

- ☐ If there is more than one inherited parameter group, click *OK* to view the next group. When you are finished viewing, close the window by clicking *Cancel*.
- ☐ If there is only one inherited parameter group, close the window by clicking *Cancel*.

Show Inherited Parameters SKILL Function

```
pcHIDisplayInheritedParameter( ) => t | nil
```

Parameterized Layer Commands

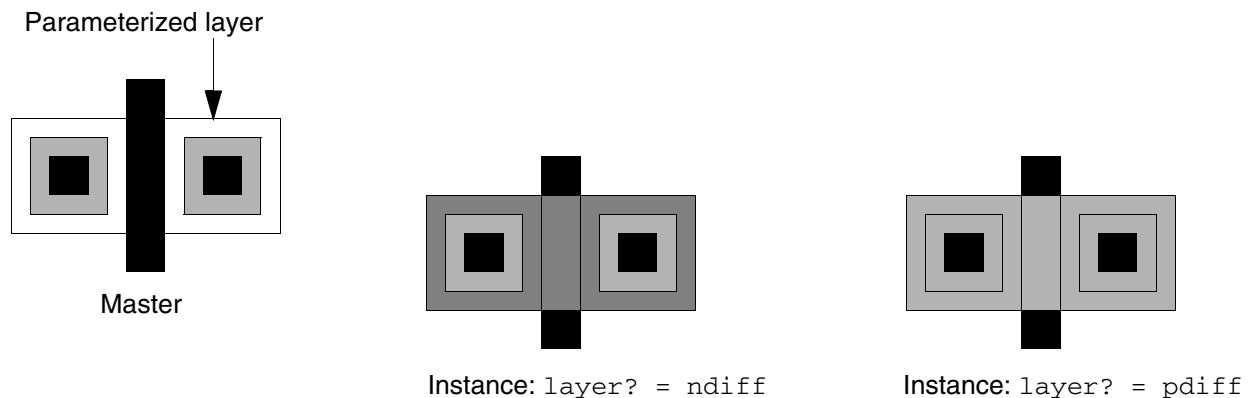
The Virtuoso® Parameterized Cell (Pcell) program lets you choose a set of shapes and associate them with a layer defined by a parameter of the Pcell.

- Assigning a Parameterized Layer to Objects
- Parameterized Layer Menu
 - ❑ Specifying an object whose layer is to be parameterized (see “Define” on page 133)
 - ❑ Changing the object or the parameter definition (see “Modify” on page 134)
 - ❑ Removing the parameterized layer parameter from an object (see “Delete” on page 134)
 - ❑ Displaying the parameter information (see “Show” on page 135)

Assigning a Parameterized Layer to Objects

1. Group objects to be assigned a parameterized layer.
2. Define the parameter for the layer.
3. (Optional) Define the purpose for the layer.

When you place an instance of the Pcell, you control the layer associated with the shapes with the value you give the parameter. If you define the parameter with the name `layer?`, the system prompts you to enter a value for the layer when you place the instance. The value can be any layer name and must be entered in the same way as the layer name is listed in the technology file. There is no default layer name. If you do not enter a layer name, the system uses the layer that was drawn in the master.



Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

Parameterized layers can also have purposes assigned to them. For example, you can assign a parameterized layer and purpose to a metal wire. The wire can be placed as metal1, metal2, or metal3, and its purpose can be either power or ground. You define these new purposes in the technology file. Later, you can display or plot layers according to their purpose.

Parameterized Layer Menu

Parameterized Layer options create a parameter that changes the layer of an object when an instance is placed. The options also allow you to specify the color and lock state of a group of shapes, simultaneously. Use *Define* to assign a layer parameter to selected objects. Use *Modify* to change the object or the parameterized layer definition. Use *Delete* to remove a parameterized layer definition from an object. Use *Show* to display the parameterized layers.

Define

Assigns a layer parameter to selected objects in a Pcell so you can change the layer of the objects when you place the Pcell.

To use Define:,

You can preselect the objects or select them after starting the command.

1. In a layout window, choose *Pcell – Parameterized Layer – Define*.

If you have not preselected any objects, the system prompts you to select the objects. The objects you select must be drawn on the same layer.

2. Click the objects you want.
3. Double-click, or press `Return/Enter`, to stop selecting.

The Define Parameterized Layer form appears.

4. In the *Layer Parameter or Expression* field, type a layer parameter or expression.

You specify either the name of a parameter (variable), such as `Metal1`, or the SKILL expression, such as `mod(maskNum+numTracks 2) + 1`.

5. (Optional) In the *Purpose Parameter or Expression* field, type a purpose parameter or expression.
6. (ICADVM18.1 only) In the *Layer Mask Value or Expression* field, type the string or integer value to specify the mask color of the shapes in the parameterized layer.
7. (ICADVM18.1 only) In the *Lock State Value or Expression* field, type the string or Boolean value to specify whether the shape's mask color is locked or not.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

8. Click *OK* and press `Escape` to end the command.

The parameter definitions you entered in the form are assigned to the selected objects.

Define Parameterized Layer SKILL Function

```
pcHIDefineLayer( ) => t | nil
```

Modify

Changes a layer parameter associated with a group of objects. Changes the objects in the group or the parameters assigned to the group. In addition, it modifies the expressions for shape color and lock state.

To use Modify:

1. In a layout window, choose *Pcell – Parameterized Layer – Modify*.

The system prompts you to select an object in the parameterized layer group.

2. Click an object in the parameterized layer group.

The system highlights all the objects in that parameterized layer group and displays the Modify Parameterized Layer form.

The values shown in this form were defined using the Define Parameterized Layer form.

If the object you select is not a member of a parameterized layer group, a warning message appears in the CIW and you cannot complete the command.

3. Change the definitions.
4. Click *OK* and press `Escape` to end the command.

Modify Parameterized Layer SKILL Function

```
pcHIModifyLayer( ) => t | nil
```

Delete

Removes a layer parameter from a group of objects.

To use Delete,

1. In a layout window, choose *Pcell – Parameterized Layer – Delete*.

The system prompts you to select an object in the parameterized layer group.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

2. Click an object in the parameterized layer group.

The system highlights all the objects in that parameterized layer group and displays the *Delete Parameterized Layer form*, showing information about the group.

The values shown in this form were defined using the *Define Parameterized Layer form*.

3. Do one of the following:

- ☐ If you do not want to delete the layer parameter for the group of objects, click *Cancel*.
- ☐ If you want to delete the layer parameter from the group of objects, click *OK*.

Delete Parameterized Layer SKILL function

```
pcHIDeleteLayer( ) => t | nil
```

Show

Highlights a group of shapes in a parameterized layer group and displays information about the group.

To use Show,

1. In a layout window, choose *Pcell – Parameterized Layer – Show*.

The system highlights the shapes in a layer parameter group and displays the Show Parameterized Layer window, showing information about the group.

2. Do one of the following:

- ☐ If there is more than one parameterized layer group, click *OK* to view the next group. When you are finished viewing, close the window by clicking *Cancel*.
- ☐ If there is only one layer parameter group, close the window by clicking *Cancel*.

Show Parameterized Layer SKILL Function

```
pcHIDisplayLayer( ) => t | nil
```

Parameterized Label Commands

- Assigning Parameterized Labels
- Using the *Parameterized Label* menu, including
 - Specifying the label you want placed on each instance of the Pcell (see “Define” on page 137)
 - Changing the location or definition of the label (see “Modify” on page 139)

Assigning Parameterized Labels

A parameterized label is not an instance name, but a label displaying values within the Pcell, such as width and length of gates. To place text that varies in cell instances, you use parameterized labels.

To assign a parameterized label to objects in your Pcell,

1. Define the text for the parameterized label.
2. Specify the origin of the text placement.
3. Assign options for rotation or justification.

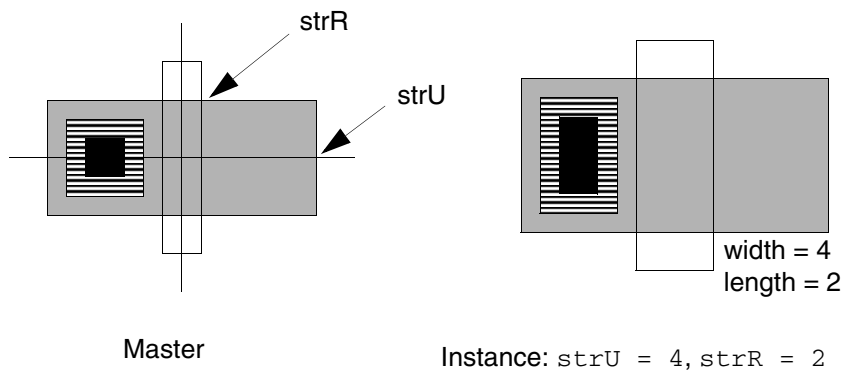
The text can vary in each instance of the Pcell, depending upon the parameter definition. If the parameter definition is a Cadence® SKILL language expression, the value of the expression is placed. For example, you might give a parameterized transistor a label showing the values for width and length, where `strU` and `strR` are the parameter names for the stretch control lines. You define this label with a SKILL expression such as:

```
sprintf(pcLabelText "width = %g length = %g" strU strR)
```


Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

where %g indicates the value to be shown as a floating-point decimal, and the values for `strU` and `strR` are used for the label.



Parameterized Label Menu

Parameterized Label commands work with a parameter that assigns a customized label to a Pcell. Use the *Parameterized Label* commands to create a customized label for each instance of the Pcell. Use *Define* to specify the label you want placed on each instance of the Pcell. Use *Modify* to change the location of the label or the text in the label.

Define

Places a parameterized label on an instance of a Pcell.

To use Define,

1. In a layout window, choose *Pcell – Parameterized Label – Define*.

The Define Parameterized Label form appears.

2. In the *Label* field, type a definition for the label.
3. Change the settings on the form to indicate any options you want to use for the label.
4. Click *OK*.

The system prompts you to specify an anchor point for your label.

5. Click to specify the anchor point for your label.
6. Press `Escape` to end the command.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

Error Conditions

When you complete the form, the system checks the definition to make sure that the label follows SKILL syntax rules. When you place an instance of the Pcell, the system evaluates the expression using the values you entered for any parameters referenced in the expression. For example, if the label is defined as

```
sprintf(pcLabelText "width=%g" width)
```

the Pcell program uses the value you enter for `width` when you place each instance.

The label expression does not have to evaluate to a string. If the expression evaluates to `nil`, no label is placed. For example, you define a label as

```
sprintf(pcLabelText "feedthrough=%d" pcIndexX)
```

with conditionally included feedthroughs. You place an instance that does not include any feedthroughs; so no label is placed in the instance.

If you reference `pcIndexX` or any of the reserved symbols for repetition groups in your parameterized label, you must include the label in the repetition group.

If the expression contains references to any symbol you have not defined as a parameter of the Pcell (for example, to control stretching or conditional inclusion), an error message appears and the compiler does not compile the cell. Define the parameterized label last, after you have defined all other parameters of the Pcell and verified the parameter definition syntax using *Parameters – Summarize*.

Using *sprintf*

The most commonly used parameterized labels build up a string using the SKILL `sprintf` function. The first argument of the `sprintf` function is the name of a symbol that stores the string. The compiler checks all parameterized labels for references to any symbols that are not parameters of the Pcell. If the Pcell program finds references to any symbols that are not parameters, the system issues an error message.

There are two ways to avoid an error message:

- Use the special symbol, `pcLabelText`, within `sprintf` functions for parameterized labels.
- Use `nil` as the first argument to `sprintf`.

Examples Using *sprintf*

```
sprintf( pcLabelText "channel width = %g" width)  
concat( "Output" pcIndexX )
```

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

```
list( param1 param2 param3 param4 )  
sprintf(nil "channel width = %g" width)
```

Define Parameterized Label SKILL Function

```
pcHIDefineLabel( ) => t | nil
```

Modify

Modifies a parameterized label you have associated with an object.

To use Modify,

1. In a layout window, choose *Pcell – Parameterized Label – Modify*.

The system prompts you to select the parameterized label you want to modify.

2. Click the parameterized label you want to modify.

The Modify Parameterized Label form appears.

The values shown in this form were defined using the *Define Parameterized Label* form.

3. Change the values for the label.
4. Click *OK* and press `Escape` to end the command.

Modify Parameterized Label SKILL Function

```
pcHIModifyLabel( ) => t | nil
```

Parameterized Property Commands

- Using Parameterized Properties
- Using the *Parameterized Property* menu, including
 - Assigning a property to the Pcell or changing its definition (see “Define/Modify” on page 141)
 - Removing the parameterized property (see “Delete” on page 141)
 - Listing all parameterized properties assigned to the Pcell (see “Show” on page 142)

Using Parameterized Properties

The values of parameterized properties are determined from a Cadence® SKILL language expression that references the parameters of the Pcell. For example, you might define a property recording the drive capability (in picofarads) of a transistor with variable width and length.

Property values are not visible, nor are they directly accessible from the layout editor. The property values are accessible only through SKILL procedural access to the master.

Using sprintf to Assign a Parameterized Property

You can use the SKILL `sprintf` function to build a string to use as the value of a parameterized property. The first argument of the `sprintf` function is the name of a symbol that stores the string. The compiler checks all parameterized properties for references to any symbols that are not parameters of the Pcell.

If the expression contains references to any symbol you have not defined as a parameter of the Pcell, an error message appears and the compiler does not compile the cell.

There are two ways to avoid getting an error message:

- Use the special symbol `pcPropText` within `sprintf` functions for parameterized properties
- Use `nil` as the first argument to `sprintf`

Examples of Parameterized Properties

```
sprintf( pcPropText "channel width = %g" width)
concat( "Output" pcIndexX )
```

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

```
list( param1 param2 param3 param4 )  
sprintf(nil "channel width = %g" width)
```

Parameterized Property Menu

Parameterized Property commands create a parameter that lets you attach properties whose values are determined from a SKILL expression. Use *Define/Modify* to create and change a parameterized property. Use *Delete* to remove parameterized properties from a Pcell. Use *Show* to list all parameterized properties assigned to a Pcell.

Define/Modify

Attaches parameterized properties to a Pcell.

To use Define/Modify

1. In a layout window, choose *Pcell – Parameterized Property – Define/Modify*.
The Parameterized Property form appears.
2. In the *Property Name* field, type the name of the property you want to define or modify.
3. In the *Name or Expression for Property* field, type an expression for the value of the property.

When you type an existing property name in the *Property Name* field, the system automatically fills in the *Name or Expression for Property* field.

4. Click *OK* to close the form.

Define Parameterized Property SKILL function

```
pcHIDefineProp( ) => t | nil
```

Delete

Deletes one parameterized property at a time from a Pcell.

To use Delete,

1. In a layout window, choose *Pcell – Parameterized Property – Delete*.

The Delete Parameterized Property form appears.

The values shown in the form were defined using the Parameterized Property form.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

2. Do one of the following:

- ☐ If you do not want to delete the property shown, click *Next*.

The next parameterized property appears in the form. When all the parameterized properties have been shown, the following message appears in the CIW:

WARNING Delete Parameterized Property: No more groups are defined

- ☐ If you want to delete the property shown, click *OK*.

3. Click *Cancel* to close the form.

Delete Parameterized Property SKILL Function

```
pcHIDeleteProp( ) => t | nil
```

Show

Displays the parameterized properties assigned in the current cellview.

To use Show,

1. In a layout window, choose *Pcell – Parameterized Property – Show*.

The Show Parameterized Property window appears, showing the parameterized properties for the current cellview.

2. To close the window, click *Cancel*.

Show Parameterized Property SKILL Function

```
pcHIdisplayProp( ) => t | nil
```

Parameters Commands

- Modifying Parameter Default Values for Pcell Instances
- Using the *Parameterized* menu, including
 - Viewing a list of all compiled parameters defined for the Pcell (see “Edit Parameters” on page 145)
 - Viewing a list of the names of all parameters assigned to the Pcell (both compiled and uncompiled) and how they are used, including the syntax of all SKILL expressions used in their definitions (see “Summarize” on page 148)

Using Parameter Commands

After you define parameters for a Pcell, you can make changes to the data type and value of the parameters. The changes you make do not affect instances of the master Pcell until you recompile.



When you override the default for the data type of a parameter, you can corrupt the internal representation of the Pcell. For example, if the parameter `deltaX` is the stepping distance in a repetition expression, do not change the data type for `deltaX` to anything other than a number or the system will issue the error message `PcellEvalFailed` when you place an instance of the Pcell.

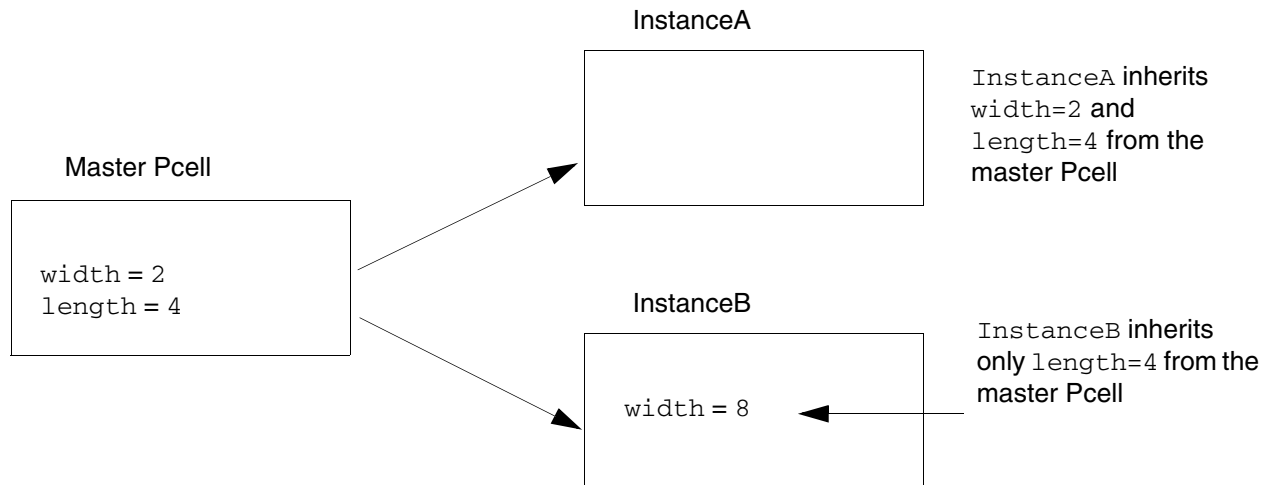
Modifying Parameter Default Values for Pcell Instances

Instances of a Pcell inherit the default values of parameters from the master Pcell. When you change the value of a parameter for an instance, the system saves the parameter and the new value with the instance; the system will not override your change, even when the parameter default value changes in the master Pcell.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

For example, for a master Pcell with `width` and `length` parameters that have default values of 2 and 4 respectively, you place two instances. For `InstanceA`, you do not change the default values. For `InstanceB`, you change the `width` parameter to 8.



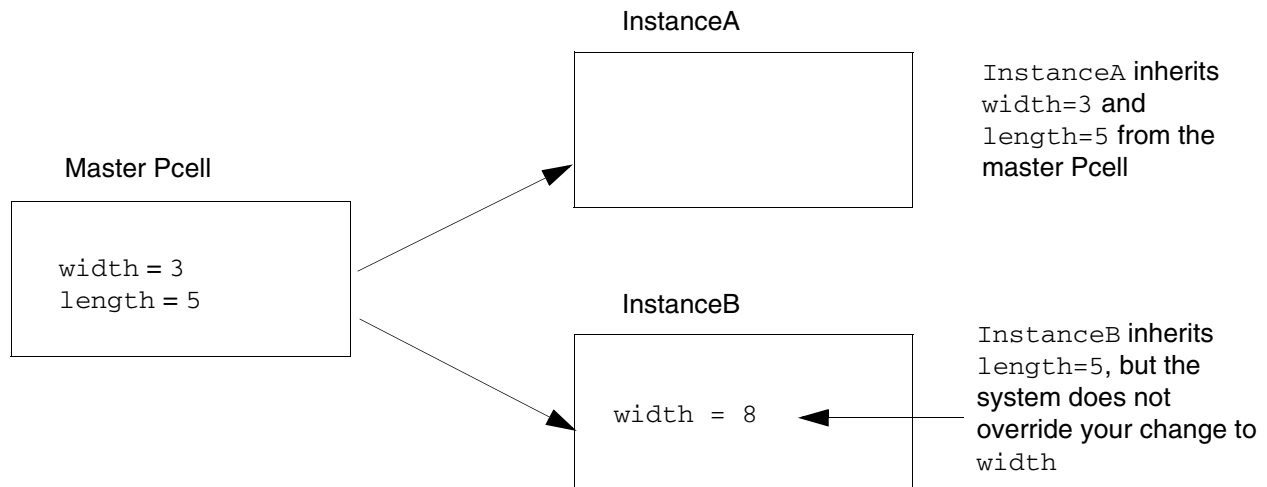
`InstanceA` inherits the default values for both the `width` and `length` parameters from the master Pcell. `InstanceB` inherits only the default value for the `length` parameter; the `width` parameter has become part of `InstanceB` and remains set to 8 until you change it.

Later, you change the master Pcell default values for the `width` and `length` parameters to 3 and 5, respectively, and recompile the Pcell. `InstanceA` automatically inherits the new values for both `width` and `length`. `InstanceB` inherits the new value for `length` only.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

InstanceB cannot inherit the new value for `width` because you (or another user) modified the value of the `width` parameter for that specific instance.



Parameters Menu

Parameters commands let you view the names, data types, values, and other information about the parameters assigned to the cellview. You can change the data type and value for any parameter. Use *Edit Parameters* to look at a list of all compiled parameters defined for the Pcell. You can change the data type and/or value for any parameter. Use *Summarize* to view a list of the names of all parameters assigned to the Pcell (both compiled and uncompiled) and how they are used, including the syntax of SKILL expressions used in their definitions.

Edit Parameters

Displays a list of all compiled parameters defined for the cellview. You can change the *Data Type* and *Value* fields for all parameters.

The changes you make on the Edit Parameters form do not affect instances of the Pcell until you recompile the Pcell.

To use Edit Parameters,

1. In a layout window, choose *Pcell – Parameters – Edit Parameters*.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

If the Pcell needs to be recompiled, a dialog box appears prompting you to recompile the Pcell. The Edit Parameters form displays only compiled parameters. Recompile the Pcell now if you want to see new parameters added since the last compile.

After you close the dialog box, if any, the Edit Parameters form appears showing the names of all compiled parameters defined for the current cellview and their data types and values. If the parameter name is highlighted, the *Data Type* or *Value* field was changed since the last compile.

Parameter Name	Data Type	Value	
M1H	Int	2	Reset to Defaults
M2H	Int	3	Reset to Defaults
paramA	Int	1	Reset to Defaults
paramB	Int	1	Reset to Defaults
paramM	String	No	Reset to Defaults
paramMdot	Boolean	False	Reset to Defaults

If you changed the *Data Type* or *Value* field for a parameter in the Edit Parameters form since the last compile, the parameter name is highlighted to show there are uncompiled changes.

paramA	Boolean	True	Reset to Defaults
paramB	Int	1	Reset to Defaults

2. Change the *Data Type* field for any of the parameters.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

When you change the data type for a parameter, the system automatically converts the data in the *Value* field to the new data type. For example, when you change a Boolean parameter to floating point (*Float*), a value of *False* is converted to 0, and a value of *True* is converted to 1.

If you change from one type to another, then immediately back to the previous data type (without changing the *Value* field), the program restores the *Value* field to its previous value.

For example, if you change the *Data Type* from *Float* to *Integer*, then back to *Float*, the program restores the original floating point number.

Data Type	Value
Float <input type="checkbox"/>	3.5

to

Data Type	Value
Int <input type="checkbox"/>	3

then back to

Data Type	Value
Float <input type="checkbox"/>	3.5

3. Change the *Value* field for any of the parameters.

If you want to change the default value for a parameter and the value you want to enter does not match the current data type, first change the *Data Type* field.

If you try to change the value so that it does not match the current data type, the system does not accept the change. When you move the cursor to another field or click *Apply* or *OK*, the system restores the *Value* field to its previous value.

4. Click *OK* to save your changes.

Edit Parameters Example

Suppose you defined a conditional inclusion parameter with the expression

```
ctype == "dffp"
```

The system sets the data type for conditional inclusion parameters to *Boolean* with a value of *True*. To change the value to the string `dffp`, first change *Data Type* to *String*, then change *Value* to `dffp`.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

Edit Parameters SKILL Functions

```
pcHIEditParameters( ) => t | nil  
pcModifyParam(d_cv s_param t_paramType g_paramExpr) => d_paramId
```

Summarize

Displays a summary of all parameters defined for the current cellview, including parameters added since the last compile. Use *Summarize* to check the names of parameters assigned to the Pcell, how they are used, and the syntax of SKILL expressions used in their definitions.

To use Summarize,

1. In a layout window, choose *Pcell – Parameters – Summarize*.

A text window appears, showing information about each parameter in the current cellview, such as the parameter name, data type, default value, stepping distance, number of repetitions, margin, and so forth. Scroll the window to see all the data.

2. Choose *File – Close Window* to close the window.

Summarize Parameters SKILL Function

```
pcHISummarizeParams( ) => t | nil
```

Compile Commands

- [Creating a Pcell from a Cellview](#)
- [Creating a SKILL File from a Cellview](#)
- [Recognizing Error Conditions](#)
- Using the *Compile* menu, including
 - Saving your layout cellview as a parameterized cell (see [“To Pcell”](#) on page 151)
 - Saving your layout cellview as a SKILL file (see [“To SKILL File”](#) on page 152)

Using Compile

The Virtuoso® Parameterized Cell (Pcell) compiler lets you create Pcells and Cadence® SKILL language files from your graphic design.

Creating a Pcell from a Cellview

The *To Pcell* command creates a Pcell in the database from the design in the current window. If you do not compile a cell before you place an instance of it in another cellview, the system interprets the design as a standard fixed cell instead of a Pcell. The Pcell must be recompiled after any changes to the layout or parameters. Otherwise, the instance you place reflects the old parameter values.

The first time you compile a Pcell, you must specify a function for the Pcell. This creates a property named `function` for the Pcell and allows you to access a section of the technology file to create specific spacing rules for layers in the Pcell. The function can be

- Transistor (default)
- Contact
- Substrate contact
- None

Creating a SKILL File from a Cellview

The *To SKILL File* command creates a SKILL file from the data in the current cellview. The SKILL file represents the data in the cell using `pc` functions. You can then edit the file like any other SKILL file. You can use the SKILL language to customize it.



Caution

When you compile a file to SKILL, you should give it a name different from the layout cellview name. This way, when you load the SKILL file back into the layout environment after you edit it in SKILL, it will not overwrite the original layout cellview.

You can load the SKILL file into the graphics environment by typing

```
load "filename.il"
```

The system creates a layout cellview from the SKILL file using the cell name you assigned on the Compile To Skill form. After you compile a cellview to a SKILL file and load it back into the graphics environment, you can edit it using only SKILL. You cannot use the layout editor to edit a cellview created from a SKILL file. The layout cellview created when you load a SKILL file into the graphics environment contains only the following message:

Warning: The supermaster is defined by the SKILL procedure associated with this cellView

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

Do not create the SKILL file in your layout editor library. These libraries are reserved for files created with the layout editor. When you give the path for the SKILL file, do not put the file in the library where the cellview is located. If you do not specify a path for the file, the compiler creates the file in the directory containing your library. If you put the file in a different directory, you will need to use the `setSkillPath` function to tell the system where to look for the file before you can load it to a cellview.

Recognizing Error Conditions

If any of the following conditions is true, the CIW displays a warning message and the compiler does not compile the cell to a Pcell or a SKILL file.

- You define a parameterized label or property expression that references a symbol you have not defined as a parameter of the cellview
- You define a repetition or conditional parameter that references a dependent stretch control line you have not defined

Compile Menu

Compile commands create a Pcell or a SKILL file from the current cellview. Use the *Compile* commands to compile your cellview to a Pcell to be placed in your designs or a SKILL file that can be customized using the SKILL programming language. Use *To Pcell* to save your layout cellview as a parameterized cell. Use *To SKILL File* to save your layout cellview as a SKILL file.

To Pcell

Creates a Pcell in the database from the design in the current window. If you do not compile a Pcell before you place an instance of it in another design, the system interprets the design as a standard fixed cell instead of a Pcell. Each time you edit the graphic Pcell, you must recompile it so that all placed instances reflect the changes.

If you make changes to a Pcell and forget to recompile, problems might occur in designs that contain instances of the Pcell. Therefore, when you change a Pcell master, do not recompile, and try to save the design, the system displays the Compile To Pcell form to remind you to recompile.

Note: Before you compile a cell, you must define all parameters for the cell.

To use To Pcell,

1. In a layout window, choose *Pcell – Compile – To Pcell*.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

If this is the first time you have compiled this Pcell, the Compile To Pcell form lets you assign the *function* property used by the Virtuoso Compactor. If you have compiled the Pcell and want to change the function property, choose Edit – Properties.

2. Select a function.

The default is *transistor*.

3. Click *OK*.

The Pcell compiler creates a master for the Pcell. The system automatically updates all instances of this master in other cellviews.

Compile to Pcell SKILL Function

```
pcHIDefineParamCell( ) => t | nil
```

To SKILL File

Creates a SKILL file from the data in the current cellview. The file can then be edited as any SKILL file.

Note: Before you compile a cell to a SKILL file, you must define all parameters for the cell.

To use To SKILL File,

1. In a layout window, choose *Pcell – Compile – To SKILL File*.

The Compile To Skill form appears.

2. In the *Library Name*, *Cell Name*, and *View Name* fields, type the names for the library, cell, and view.
3. In the *File Name* field, type the path and a filename.

If you do not specify a path, the software creates the file in the first writable directory in your SKILL path, usually *your_install_dir/dfII/local*.

4. Click *OK*.

The Pcell compiler writes all information in the current cellview to a SKILL file. You can now view and edit the file with any text editor.

For more information about using the SKILL programming language, refer to *Cadence SKILL Language User Guide*.

Related Topics

[pcHIDefineParamCell](#)

[pcDefinePCell](#)

Compile to SKILL File SKILL Function

```
pcHICompileToSkill( ) => t | nil
```

Recompile Pcells

If a design is compiled into Pcell successfully, then the `_pcCompiledCounter` property is added to the Pcell. The value of the `_pcCompiledCounter` property of the Pcell is equal to the current value of `modifiedCounter` of the Pcell.

Therefore, when the value of the `modifiedCounter` of the Pcell is greater than the value of `_pcCompiledCounter` of the Pcell, it means that the Pcell has to be modified after it been compiled as a Pcell.

Note: Only graphical Pcell contains the `_pcCompiledCounter` property.

Make Ultra Pcell Command

- [What Are Ultra Pcells?](#)
- [Ultra Pcell Requirements](#)
- [Make Ultra Pcell](#)

What Are Ultra Pcells?

Ultra Pcells are created by compiling multiple layouts into one Pcell. During the combination, you define a selector parameter. The value of this selector parameter is then used to determine which Pcell layout to use.

You use ultra Pcells when you cannot create a single Pcell. Instead, you create two or more Pcells and build them into an ultra Pcell.

Ultra Pcells are handy when logic versus layout mapping requires a one-to-one mapping of logic cells to layout cells. The final layout cells all have the same library, cell, and view names even though they started out with different cell or view names.

The following is a good example of an ultra Pcell because the four layouts do not lend themselves to standard Pcell definition. You could implement these layouts with stretch and repeat commands, but it would be more difficult.

Ultra Pcell Example

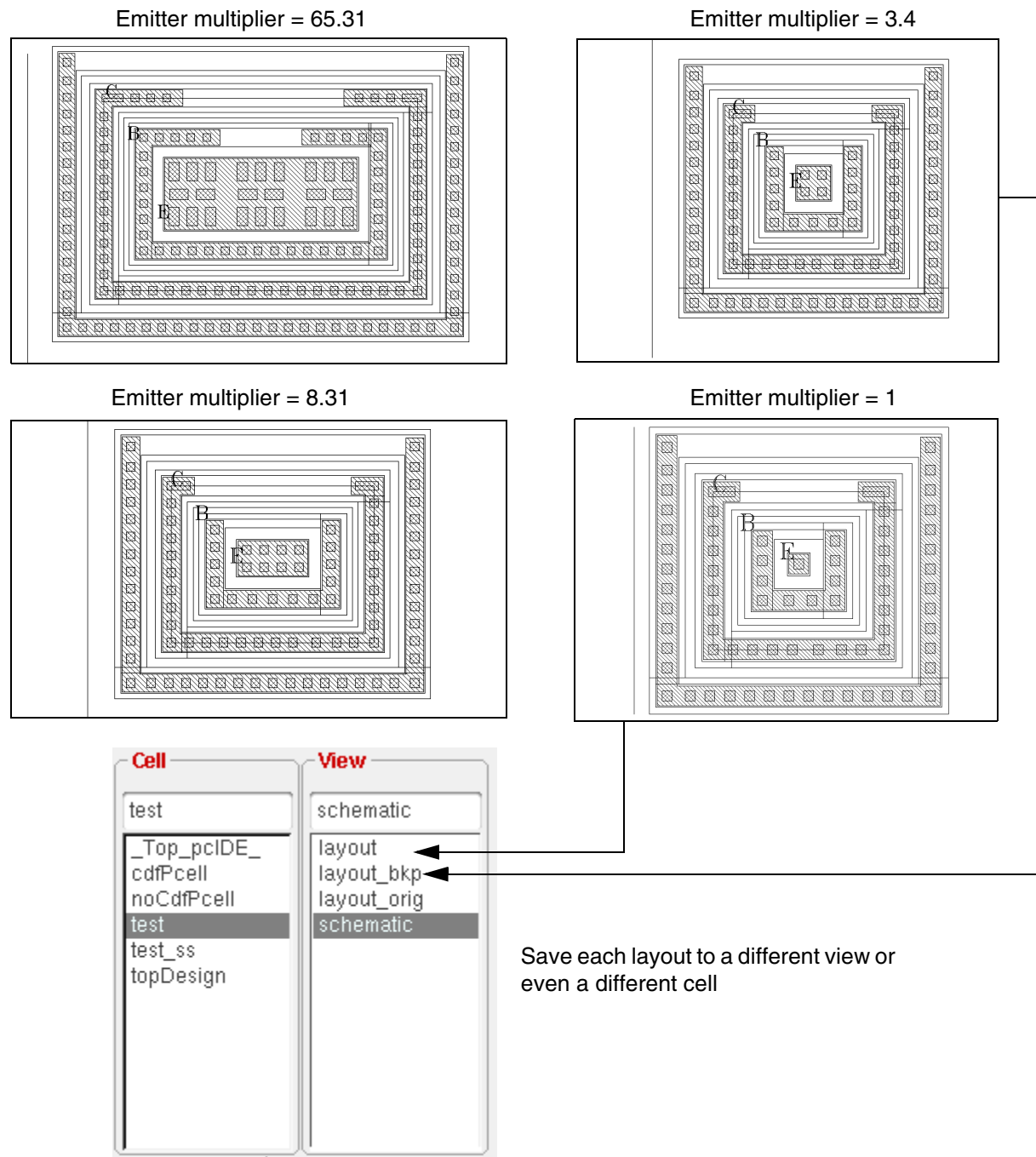
Suppose you were working on a BICMOS design where there are four versions of the *pn*p transistor and your models support only those four versions. You want to make sure that the layout uses same those four versions and no other.

1. Create, compile, and save the following four Pcells.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

Figure 6-1 pnp Transistors

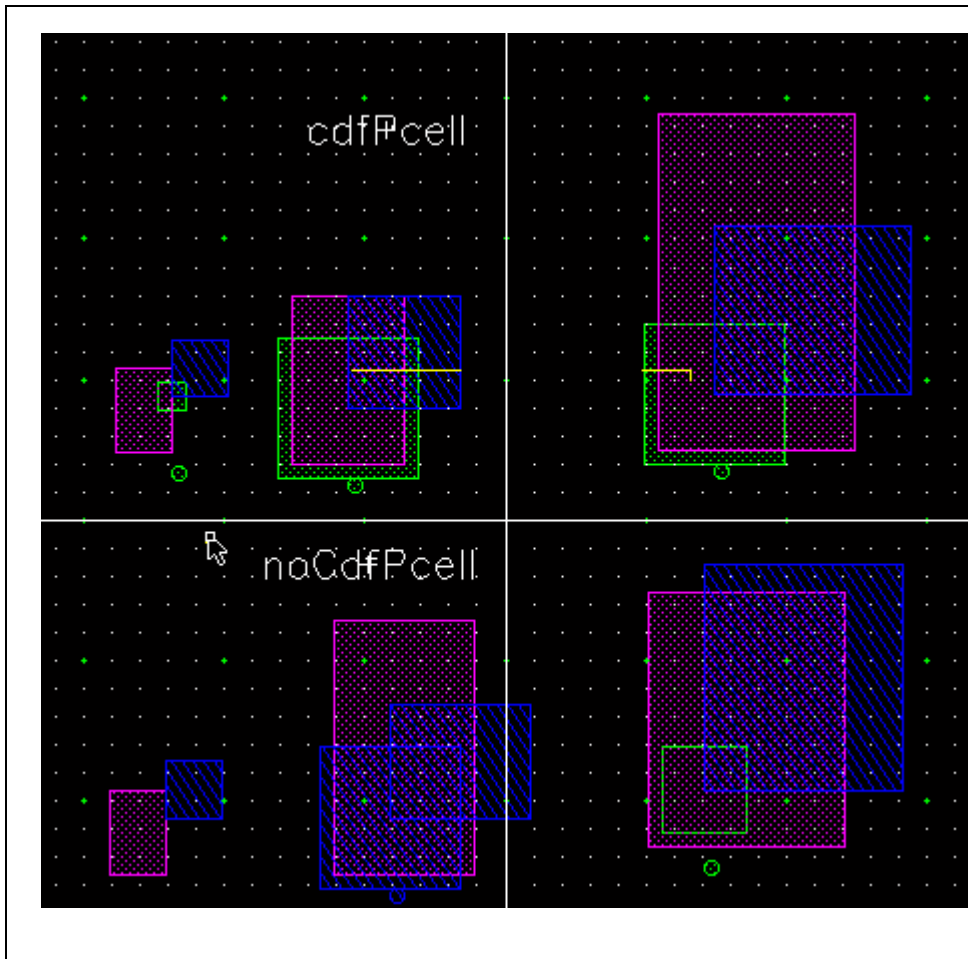


Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

2. Update the CDF (component description format) with the selector parameter (multiplier) and the selector values (emitter multiplier).
3. Compile the Pcells into an ultra Pcell.

You can now use the Create Instance form to choose the correct layout by selecting the value of the multiplier. The appropriate layout is automatically generated.



Ultra Pcell Requirements

Ultra Pcells have the following requirements:

- Multiple Pcell layouts need not be in the same cell although they must be in the same library as the completed ultra Pcell.
- Each Pcell must compile successfully.

Virtuoso Parameterized Cell Reference

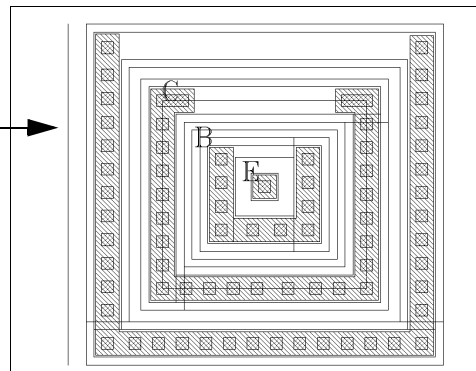
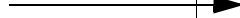
Creating Graphical Parameterized Cells

- Each Pcell must have identical parameters and default values. No Pcell can have more or fewer or different parameters from the other Pcells. When the ultra Pcell is compiled, it has all of the parameters, which work as designed in each Pcell.
- The selector parameter must be unique. It cannot match any other parameter.
- *type*, *objType*, *name*, *cell*, and *status* are reserved words and cannot be used as selector parameters.

Note: Hierarchical copy doesn't copy subcells in UltraPcell. You have to manually copy the subcells into the new library. However, you do not have to update references to the ultra pcell.

You can use a rigid (non parameterized) cell by changing it to a Pcell by adding a parameter that you do not use. For example, the cells in [Figure 6-1](#) on page 155 are all rigid layout cells. Each layout had one stretch line added that was never changed.

Stretch line: parameter = test



Make Ultra Pcell

Combines several Pcells into one Pcell.

To use Make Ultra Pcell,

1. Create Pcell layouts.

Be sure that your Pcell meets all the requirements listed in [“Ultra Pcell Requirements”](#) on page 156.

2. To update the CDF, in the CIW, choose *Tools – CDF – Edit*.

The Edit Component CDF form appears.

3. Set the *CDF Type* cyclic field to *User*.
4. Type the library and cell names and press the **Tab** key.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

The form expands.

5. Click *Add* under *Component Parameters*.

The Add CDF Parameter form appears.

6. In the Add CDF Parameter form,
 - a. Set the *paramType* cyclic field to *cyclic*.
 - b. Set the *parseAsCEL* cyclic field to *no*.
 - c. Set the *storeDefault* cyclic field to *yes*.
 - d. In the *prompt* field, type the selector parameter.

Write down your entry: you will need to type it again in the Ultra Pcell form.

- e. In the *choices* field, type the parameter values.

Write down your entry: you will need to type it again in the Ultra Pcell form.

- f. In the *defValue* field, type the default value.
 - g. Click *OK*.
7. In the Edit Component CDF form, click *OK*.
8. Compile the Pcells into an ultra Pcell by choosing *Pcell – Make Ultra Pcell* in the cellview.

The Ultra Pcell form appears.

9. In the *Selector Parameter* field, type the same value you typed in the *prompt* field in the Add CDF Parameter form (entered in step 6).
10. Type the library, cell, and view names.
11. Click *Add Cell*.

The form expands.

12. In the *Selector Value* field, type a parameter value that corresponds to one in the *choices* field in the Add CDF Parameter form (entered in step 6).
13. Repeat step 11 and step 12 as many times as you need to accommodate all the Pcells.

The form is updated to show you as many cells as you request.
14. Click *OK*.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

The ultra Pcell is compiled.

Ultra Pcell SKILL Function

```
auHiUltraPCell( )
```

Customizing the Pcell Compiler

This section is intended for advanced users only.

Customizing the Pcell compiler lets you generate from your design a Pcell that differs from the Pcell normally generated by the compiler.

You customize the compiler by writing SKILL procedures to process any objects in your design, either instead of the processing that the compiler applies to the object or in addition to it. The compiler calls your SKILL procedures as it is generating a Pcell from your design.

Background

At the database level, the master Pcell of a Pcell is represented as a SKILL procedure. When you place an instance of that Pcell with a particular set of parameter values, the database creates a temporary submaster Pcell and then executes the procedure. The procedure normally generates new objects such as shapes, instances, and terminals within the submaster Pcell.

The resulting composition of the submaster Pcell depends on both the SKILL procedure and the parameter values associated with this particular submaster. The submaster Pcell is then used as the master of the instance as if it were a standard fixed Pcell.

If you are not customizing the compiler, these details are invisible to you. The Pcell compiler automatically generates the SKILL procedure for the master Pcell from your design. It does so by traversing your design and, for each object in the design, generating a call to a SKILL procedure that, when executed, recreates that object in the submaster Pcell. It takes into account any parameter directives (such as stretches) that might apply to the object.

How Customization Works

When you write SKILL procedures to customize the Pcell compiler, your goal is to generate code for inclusion in the SKILL representation of your parameterized design. The compiler then associates your custom procedures with the master Pcell. The effect of the code you generate is not apparent until an instance of the Pcell is placed and the database evaluates the associated SKILL procedure.

Note: To customize the Pcell compiler, you must set the SKILL variable `pcGlobal.userExtend` to `t` before calling the compiler.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

You must specify customization procedures with predefined names. There is a different procedure name for each different type of object or class of objects processed by the compiler.

The predefined names and argument specifications understood by the compiler are listed below:

```
pcUserPreProcessCellView(d_cv t_tag p_port)
pcUserPostProcessCellView(d_cv t_tag p_port)
pcUserInitRepeat(l_stepX l_stepY l_repeatX l_repeatY p_port)
pcUserAdjustParameters(p_port)
pcUserGenerateProperty(d_object d_prop t_tag p_port)
pcUserGenerateTerminal(d_terminal p_port)
pcUserGeneratePin(d_pin p_port)
pcUserSetTermNetName(d_pinInst p_port)
pcUserGenerateInstancesOfMaster(d_masterCV l_instanceList t_tag p_port)
pcUserGenerateInstance(d_instance t_master p_port)
pcUserGenerateArray(d_arrayInst t_master p_port)
pcUserGenerateLPP(d_layerPurposePairHeader p_port)
pcUserGenerateShape(d_shape p_port)
pcUserPostProcessObject(d_obj t_tag p_port)
```

As the compiler traverses your design, it checks to see whether a procedure with the appropriate predefined name has been loaded and, if so, calls that procedure. Depending on the value returned (`nil` or not `nil`), the compiler does or does not generate code for the object or class of objects currently being processed. This mechanism lets you give special treatment to one particular object in your design.

Prior to compilation, you can attach a property to the object to make it readily identifiable. Then in your customization procedure, you can check for that property. When you recognize it, you can generate special code for the object and return a value other than `nil`. For objects without the identifying property, your customization procedure can just return `nil`, thereby letting the compiler process the object normally.

Within your customization procedure, you can use any of the Pcell utility functions described in the Pcell section of [Virtuoso Parameterized Cell SKILL Reference](#), such as `pcGetRepeatDefn`.

Variables

In the code you generate as output of a customization procedure, you can access the values of any of the parameters of the Pcell by referencing a SKILL variable with the same name as the parameter. For example, if the Pcell has a parameter called `width`, you can generate code in the form

```
if( width < 2.5 then ... )
```

The Pcell compiler makes available two variables, `pcLib` and `pcCellView`. You can use these variables in the code your customization procedures generate to create database

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

objects. The value of `pcCellView` is the object ID of the submaster Pcell created by the database prior to execution of the Pcell SKILL procedure.

To create objects in a submaster Pcell, use `pcCellView` in calls to `dbCreateRect` or to similar functions. The `pcLib` variable holds the object ID of the library in which the submaster Pcell is created. Use the `pcLib` variable in any database function call that accesses technology file information, such as `techGetSpacingRule`.

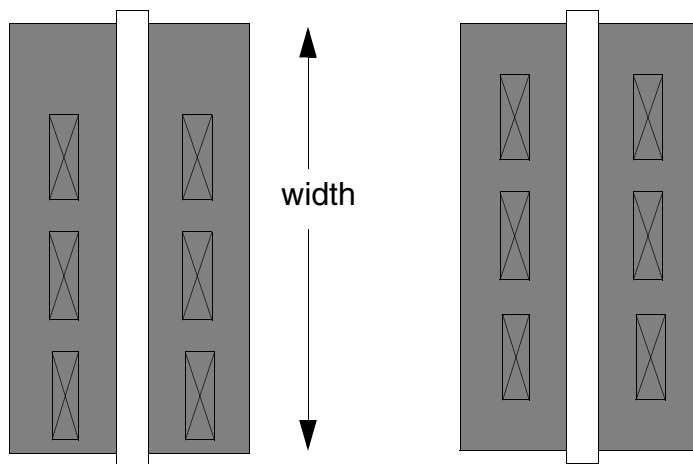
Restrictions on SKILL Functions

You can use most SKILL functions when creating your Pcell. The only functions you cannot use are `hi`, `ge`, and application-specific functions.

The program does not check the code generated by user-provided procedures. It is your responsibility to guarantee that the SKILL expressions created by your customization of the Pcell compiler are valid in all applications that access the Cadence database.

Example

The following SKILL procedures transform the Pcell on the left, which is the default output of the Pcell compiler, into the Pcell on the right. The parameter `width` controls the gate width.



The first procedure is called before the compiler processes any objects. The procedure generates code to initialize a variable:

```
procedure (pcUserPreProcessCellView( d_cv t_tag p_port )
; Initialize list to store the contacts
fprintf(p_port "PCUserContacts = nil\n")
)
```

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

The second procedure is called before the compiler processes the repeated objects. The procedure generates code to record the stepping distance and the number of repetitions:

```
procedure( pcUserInitRepeat( l_stepX l_stepY l_repeatX l_repeatY p_port )
; keep record of step distance and repetitions
fprintf(p_port "PCUserStep = %L\n" l_stepX)
fprintf(p_port "PCUserRepeat = %L\n" l_repeatY)
)
```

The third procedure is called by the compiler for each object in the Pcell after the procedure generates the code for the object. The procedure generates code to build a list of the repeated objects (contacts):

```
procedure( pcUserPostProcessObject( d_obj t_tag p_port )
if( pcGetRepeatDefn(d_obj) then
; build list of all the contact shapes
fprintf(p_port "PCUserContacts = cons(%s
PCUserContacts)\n" t_tag)
)
)
```

This last procedure is called by the compiler after the procedure has processed all the objects in the Pcell. The procedure generates code to adjust the contact spacing:

```
procedure( pcUserPostProcessCellView( d_cv t_tag p_port )
; adjust contacts up by half "slop" amount
fprintf(p_port "foreach( contact PCUserContacts \n")
fprintf(p_port "      dbMoveShape(contact d_cellView
list( 0:(width - PCUserRepeat*PCUserStep) / 2 \"R0\") )\n")
fprintf(p_port ") \n")
)
```

Pcell Compiler Customization SKILL Functions

The following lists the Pcell SKILL functions used to customize the compiler.

[pcUserAdjustParameters](#)

[pcUserGenerateArray](#)

[pcUserGenerateInstance](#)

[pcUserGenerateInstancesOfMaster](#)

[pcUserGenerateLPP](#)

[pcUserGeneratePin](#)

[pcUserGenerateShape](#)

[pcUserGenerateTerminal](#)

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

pcUserInitRepeat

pcUserPostProcessCellView

pcUserPostProcessObject

pcUserPreProcessCellView

pcUserSetTermNetName

Creating Complex Pcells

Before creating more complex Pcells, you should be familiar with graphical Pcells, your technology file, and the Cadence® SKILL language. The section discusses the following:

- Using Technology File Information

When you define an expression for a Pcell, you can retrieve values from the technology file associated with your library. Using values from the technology file lets you use the same Pcell with multiple libraries and more than one technology.

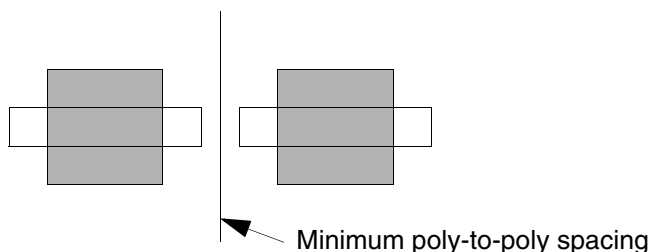
- Using the Component Description Format

When you want all the cellviews of a cell to share the same parameters, you can use CDF to define the parameters.

Using Technology File Information

When you define expressions in a graphical Pcell, you can compute values or retrieve them from the technology file associated with your library. This allows you to create expressions with values based on technology rules. Relying on values from the technology file is particularly useful when you change technologies.

For example, if you want the spacing between two objects to be minimal, you can draw a stretch control line between the two objects and give the line the minimum spacing value defined in the technology file. If the technology file changes, the separation between the objects also changes to reflect the new legal minimum separation.



Specifying Values from the Technology File

To cause a parameter to take its value from the associated technology file, you can use the `pcTechFile` function when you define the parameter. The `pcTechFile` function can evaluate SKILL expressions composed of basic SKILL constructs (with no application prefix), database functions, and technology file functions.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells



Caution

Do not use functions with application prefixes, such as `le` or `ge`. The Pcell evaluator does not understand application-specific calls, including functions with the `pc` prefix.

The syntax for `pcTechFile` is

```
pcTechFile( basic_SKILL_expression )
```

For example, the following expression retrieves the minimum spacing for poly from the technology file associated with the cellview `pcCellView` and adds it to the `length` parameter.

```
pcTechFile(
    length
    + techGetSpacingRule(
        techGetTechFile(pcCellView)
        "minSpacing"
        "poly"
    )
)
```

Database functions begin with `db` or `dd` and are documented in [Virtuoso Design Environment SKILL Reference](#).

Using Technology File Functions

When you use a database function to access technology file information, it is best to access the technology file from the Pcell itself, using `pcCellView`.

In the following example, to define a stretch control line, you must enter a value in the *Name or Expression for Stretch* field. To use the technology file value for the minimum separation between the two poly gates, you enter the following:

```
techGetSpacingRule(
    techGetTechFile(pcCellView)
    "minSpacing"
    "poly"
)
```

where `techGetTechFile(pcCellView)` returns the database identifier (dbID) of the technology file attached to the cellview.

To define the minimum separation between objects on different layers, you enter this:

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

```
techGetSpacingRule (
    techGetTechFile (pcCellView)
    "minSpacing"
    list ("poly" "drawing")
    list ("ndif" "drawing")
)
```

You can use a variety of SKILL functions to access information from the technology file. Refer to [*Virtuoso Technology Data SKILL Reference*](#) for information.

Using the Component Description Format

The component description format (CDF) provides a standard method for describing components. You can use CDF to describe parameters and attributes of parameters for individual components and libraries of components. A component-specific data item is treated similar to a parameter. Creating CDFs allows you to store all component-specific data required for a cell or a library in one location. CDF data is independent of application or view type.

If you want all cellviews of a cell to share the same parameters, use CDF to define the parameters. The CDFs apply to all views of the cell: layout, schematic, symbolic, abstract, extracted, and so forth. You can also assign parameters to a library of cells so that they are shared by all views of all cells in the library.

Note: You do not have to define a CDF unless the parameter is shared by more than one cellview.

If you want parameters on a particular cellview, such as the layout cellview, but you do not need parameters (or the same parameters) on the other cellviews, you do not need to use CDF to define the parameters.

You use component data for

- Assigning parameter names and values
- Specifying units and default values
- Verifying the valid range for values
- Dynamically changing how parameters are displayed depending on predefined conditions
- Executing a SKILL callback function whenever certain information changes
- Placing and editing components
- Netlisting and simulating

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

- Generating and synthesizing layout
- Creating backannotation
- Extracting and comparing layout to schematic
- Synthesizing logic
- Testing

Using Callbacks

CDF uses callbacks, which consist of one or more SKILL expressions or procedures. You can use callbacks with Pcells to:

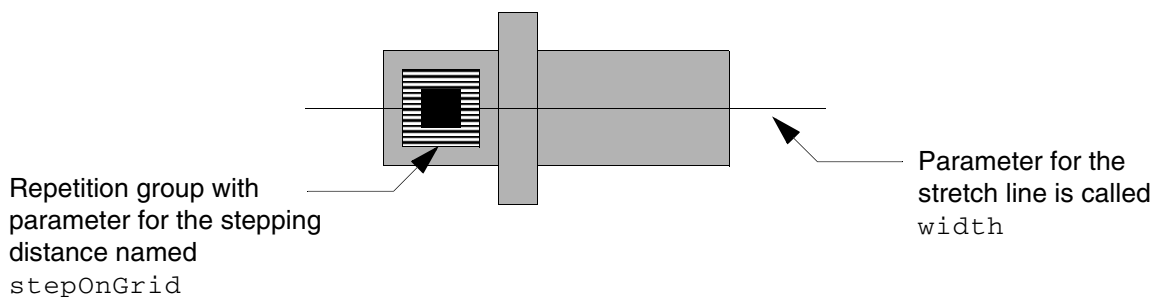
- Update the fields in the Create Instance form when you place a Pcell
- Adjust values entered in the Create Instance form
- Check special conditions you have specified and to warn you when these conditions are not met

CDF procedures do not compile into the Pcell itself, so ensure that your Pcell procedures are always defined whenever your Pcell is used. In other words, you must load the source file for the procedures before you open the library containing the Pcell.

For more information about using procedures within Pcells, see [“Variable Scoping”](#) on page 34.

Example of Pcell with CDF

In this example, your Pcell contains a parameter for the stretch line named `width` and a repetition group containing a contact with a parameter for the stepping distance named `stepOnGrid`.



Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

You can create CDF definitions for the two parameters, `width` and `stepOnGrid`, as follows:

1. Define a CDF named `stepOnGrid` to hold the value for the stepping distance.
2. Define a second CDF named `width` with
 - ❑ A prompt named *Pcell width* for the Create Instance form
The value for CDF `width` is automatically mapped to the Pcell parameter `Pcell width`.
 - ❑ A callback named `stepCalc` that uses the value of `Pcell width` to calculate the stepping distance `stepOnGrid`
3. Use a text editor to create an ASCII file named `stepCalc.il`.
 - ❑ In `stepCalc.il`, use `width` from the CDF to calculate the value for `stepOnGrid`.

Your `stepCalc.il` callback would look like this:

```
cdfwidth = cdfgData->width->value
```

```
.  
.   
.
```

```
cdfgData->stepOnGrid->value = cdfwidth/2.0
```

Retrieve the value of `width` from the CDF and store it in the local variable `cdfwidth`.

Other calculations you might want go here.

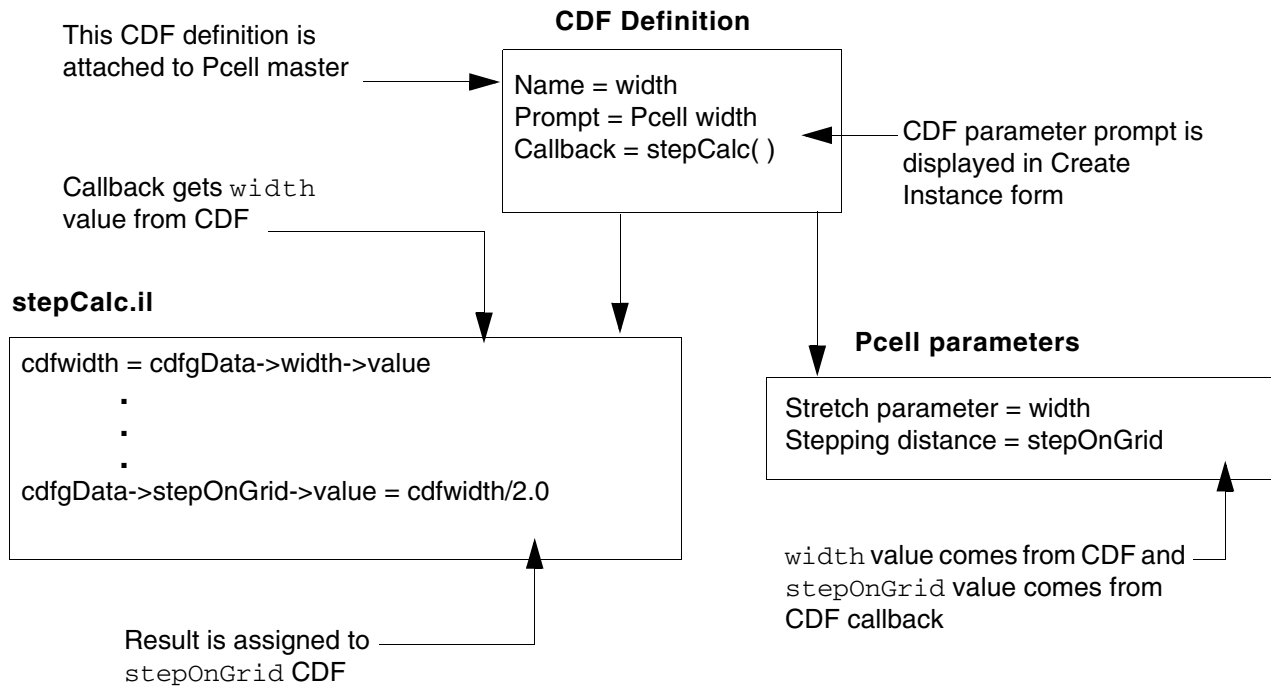
Divide the local variable `cdfwidth` by 2 and assign the result to `stepOnGrid`.

When you place an instance of the Pcell, the CDF prompt *Pcell width* appears in the Create Instance form. The stretch parameter `width` in the Pcell takes its value from the CDF `width`.

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

The repetition parameter `stepOnGrid` takes its value from the CDF callback. See the following diagram.



Refer to *[Component Description Format User Guide](#)* for more information.

Converting Graphical Pcells to SKILL Code

The *Pcell – Compile – To SKILL File* command creates a SKILL source code file from a graphically defined Pcell. It is possible to modify that SKILL code and then load it to compile a new Pcell.

While this generated code is compatible with everything described in this section, it is not optimal code because it is automatically generated from graphical input. You can, in all cases, create shorter, and much more readable, source code by starting from scratch using the methods described in this section.

One way you can use the *To SKILL File* command is to dump the code that creates a complex polygon. But there is another way to do this that might be easier. Suppose your Pcell includes a complex polygon, and you do not want to calculate and code the coordinates for this shape manually. You could use the *To SKILL File* command to dump the code that creates the complex polygon.

A quick way to get the SKILL code is to:

1. Digitize the shape in the layout editor using a temporary cellview.
2. Select the polygon.
3. In the CIW, type

```
car( geGetSelectedSet() )~>points
```

A list of output coordinates is displayed in the CIW.

4. Edit the DFII log file as follows.

The name of the log file appears at the top of the CIW. It is usually `~/CDS.log`.

- ☐ Go to the bottom of the `.log` file.
- ☐ Cut and paste the list of coordinates into your Pcell source code file.
- ☐ Delete the `\t` from the start of each line in the `.log` file.
- ☐ Place a single quotation mark in front of the list.

The code to create a polygon looks like the following:

```
dbCreatePolygon(  
    pcCellView  
    '("metall" "drawing")  
    (  
        (2.0 -348.0)  
        (144.0 -490.0)  
        (-45.0 -679.0)  
        (-333.0 -679.0)
```

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

```
)  
    )  
    (-333.0 -348.0)  
)
```

Note: While it is possible to convert a graphically defined Pcell into a SKILL-defined Pcell using the *Pcell – Compile – To SKILL File* command, it is not possible to convert a SKILL Pcell back into a graphically defined one.

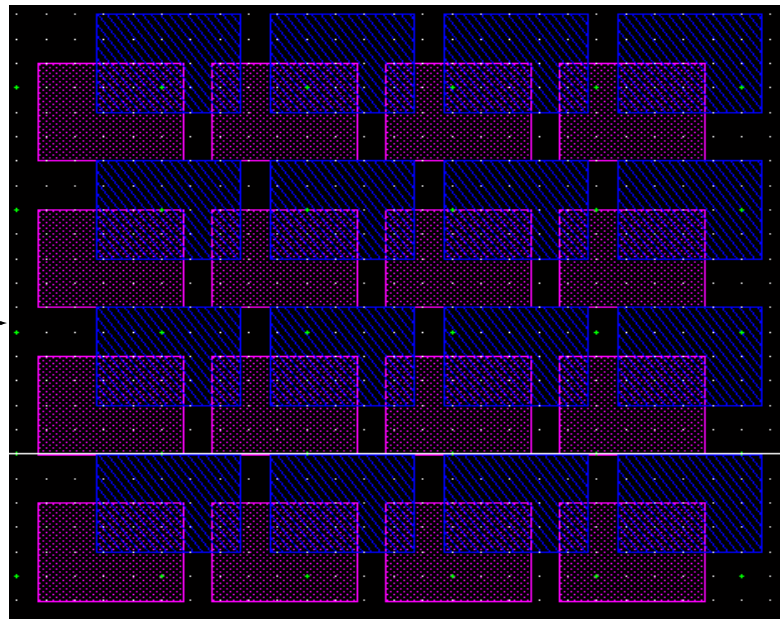
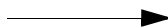
Examples of Converting Pcells to SKILL Code

The following examples of Pcells show both the graphical layout view and the SKILL created with the *To SKILL File* command.

Example 1: A Thin-Film Capacitor

The input parameters `pcW` and `pcH` define the width and height of the top metal rectangle. The rectangles on the other layers are calculated from these parameters and minimum dimension data accessed from the technology file.

Layout view



Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

SKILL for Thin-Film Capacitor Pcell

```
pcDefinePCell(
  list( ddGetObj( "pSample") "cap" "layout")
  ( ( pcW 100. )
    ( pcH 100. ) )
  let( (h2 tf 01 02 fig1 fig2 net trm pin)
;   determine some internal dimensions using data from the techFile:
    h2 = pcH * .5
    tf = techGetTechFile(ddGetObj(pcCellView))
    o1 = techGetOrderedSpacingRule(tf "minEnclosure" "metal1" "metal2")
    o2 = techGetOrderedSpacingRule(tf "minEnclosure" "metal2" "abVia")
;   Create the shapes that form the cap; metall and metal2 are pins:
    fig1 = dbCreateRect( pcCellView '( "metall1" "drawing")
      list( 0.:-h2-o1 pcW+o1+o1:h2+o1))
    fig2 = dbCreateRect( pcCellView '( "metal2" "drawing")
      list( o1:-h2 pcW+o1:h2))
    dbCreateRect( pcCellView '( "abVia" "drawing")
      list( o1+o2:o2-h2 pcW+o1-o2:h2-o2))
;   Create the nets and terminals and bind the metal pins to them:
    net = dbCreateNet( pcCellView "n1")
    trm = dbCreateTerm( net "n1" "inputOutput")
    pin = dbCreatePin( net fig1 "n1")
    pin~>accessDir = '( "left" "right" "top" "bottom")
    net = dbCreateNet( pcCellView "n2")
    trm = dbCreateTerm( net "n2" "inputOutput")
    pin = dbCreatePin( net fig2 "n2")
    pin~>accessDir = '( "left" "right" "top" "bottom")
;   Create the instNamePrefix property
    dbReplaceProp(pcCellView "instNamePrefix" "string" "cap")
    t
  ))
```

Example 2: Pcells within a Pcell

This example is a thin-film resistor that can be implemented using three different resistive media. Three graphical Pcells were created, one for each type of resistor.

- Each of these Pcells can be stretched in length and width.
- A single cell named `res` lets you switch between the three types.
- The SKILL Pcell places an instance of the specified graphical Pcell, then sets its properties to match the length and width parameters.
- The individual resistor Pcells themselves can be defined with SKILL.

This example also shows how the pins on graphical Pcells are a level down in the hierarchy and how to copy pins from instances up into the Pcell cellview. If you omit this step, the

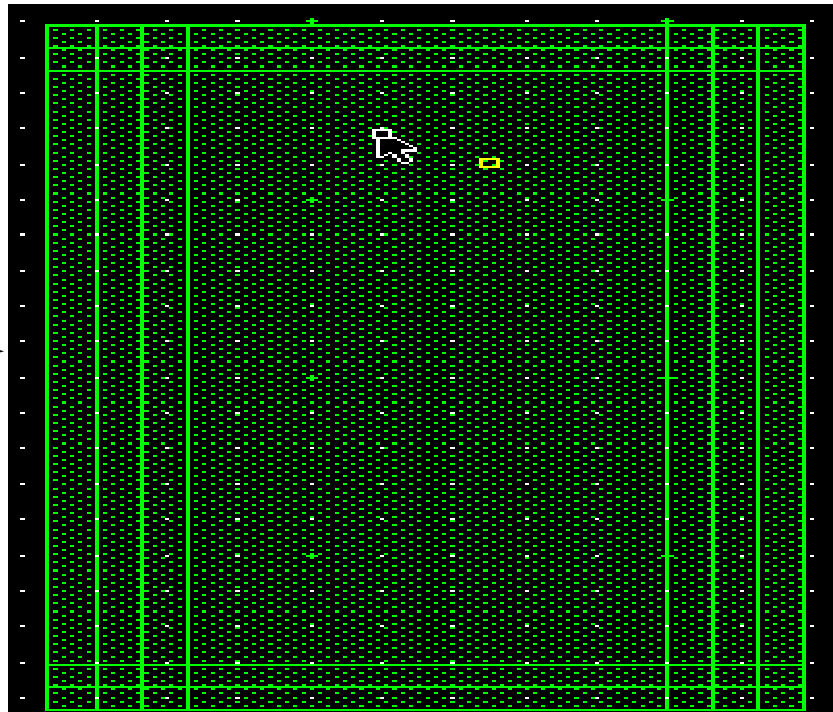
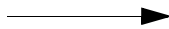
Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

sample cell has no pins. In this case, assume that the graphical Pcells have the same pin names as the `res` component.

The cell in this example closely approximates the behavior of ultra Pcells. For more information about ultra Pcells, refer to [Make Ultra Pcell Command](#).

Layout view



SKILL for Thin-Film Resistor

```
pcDefinePCell(  
  list( ddGetObj("pSample") "res" "layout")  
  ( (w_pc 20.)  
    (l_pc 37.)  
    (pcellName "resNiCr") )  
  let( lib master inst subMaster newnet term fig newfig newpin)  
;  
  place the specified PCell instance:  
  lib = pcCellView~>lib  
  master = dbOpenCellViewByType( lib~>name pcellName "layout")  
  inst = dbCreateParamInst( pcCellView master nil 0:0 "R0" 1  
    list( list("w_pc" "float" w_pc) list( "l_pc" "float" l_pc)))  
;  
  copy all layout pins in cell just placed to the PCell cellview:  
;  
  (this block of code can be copied and used in any similar PCell)  
  subMaster = inst~>master  
  foreach( net subMaster~>nets  
    newnet= dbCreateNet( pcCellView net~>name)
```

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

```
term = net~>term
dbCreateTerm( newnet net~>name term~>direction)
foreach( pin term~>pins
  fig = pin~>fig
  newfig= dbCreateRect( pcCellView fig~>lpp
    fig~>bBox)
  newpin= dbCreatePin( newnet newfig pin~>name)
  newpin~>accessDir = pin~>accessDir
)
)
; Add the 'instNamePrefix' property.
dbReplaceProp(pcCellView "instNamePrefix" "string" "res")
t
))
```

Example 3: Calling User-Defined Procedures

This example shows the use of external called procedures. The cell is a circular spiral inductor defined by outer diameter, inner diameter, line width, and spacing. The layer number on which it is built is also passed into the Pcell as an integer.

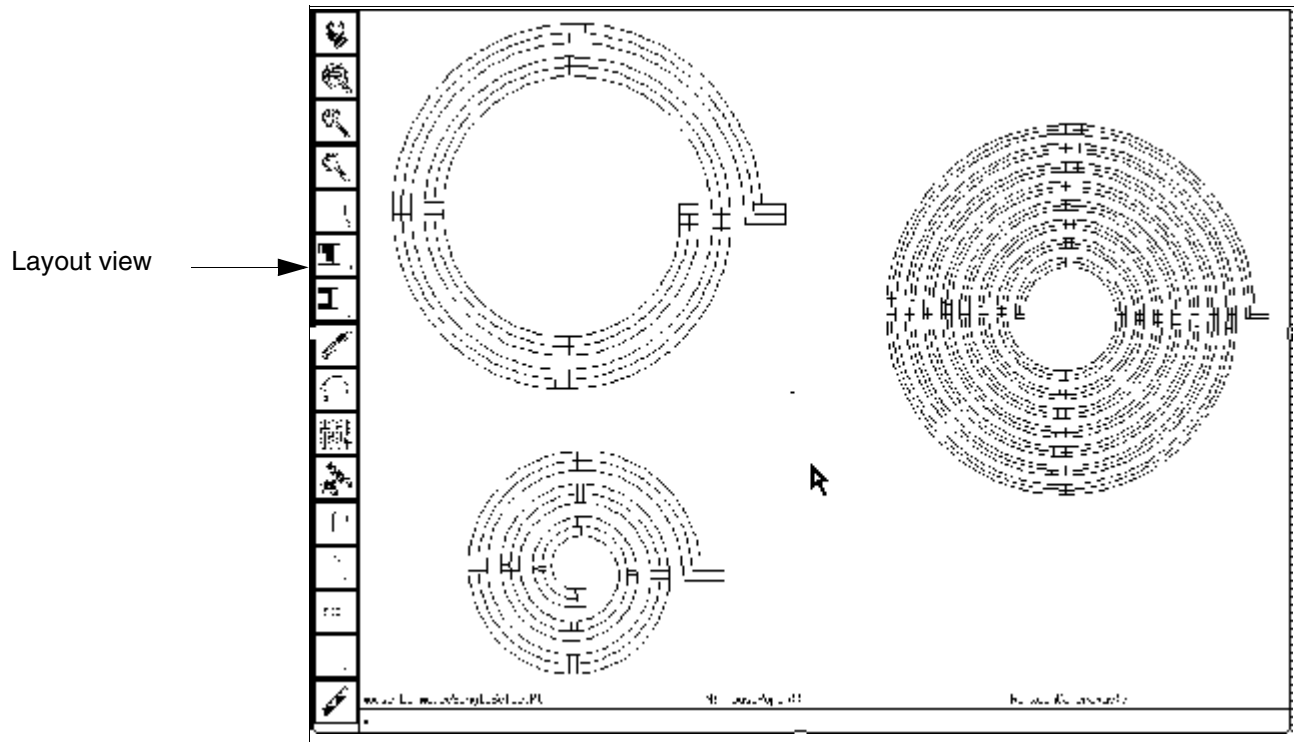
To avoid limitations on the number of points in paths when the design is converted to GDSII, each quarter turn of the spiral is built with its own path shape. The path shapes do not overlap at their ends, so they form a continuous spiral of metal.

The example shows the two procedures called from within the Pcell. The first procedure returns a list of path point lists that form the spiral. The second procedure creates a net,

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

terminal, and pin given a net name, the figure that will be the pin, and the access direction for the pin.



Procedures Called from Thin-Film Resistor Pcell

```
procedure( uwlSpiralPaths( inner outer width space nc)
  let( ( n nq ns nx xo x y chCos chSin c r twopi d pointLists points )

;   compute the number of turns:
  n = (outer - inner) / 2. / (width + space)

;   compute number of quarter turns and sides per quarter turn:
  nq = fix( n * 4)
  ns = fix( nc / 4)

;   calculate some intermediate parameters:
  twopi = 2 * 3.1415926
  d = twopi / nc           ;nc is number of chords per 360 degrees
  chCos = cos( d)
  chSin = sin( d)
  c = (width + space) / nc
  r = outer * .5

;   build the point lists:
  pointLists = nil
  x = r
  y = 0.
  points = list( x:y x+width+space:y)
  for( i 1 nq
    nx = ns
```


Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

```
for( j 1 nx
  xo = x
  x = (chCos * xo - chSin * y) * (1. - c / r)
  y = (chCos * y + chSin * xo) * (1. - c / r)
  points = cons( x:y points)
  r = r - c
)
pointLists = cons( points pointLists)
points = list( car( points) cadr( points))
)
pointLists ;the procedure returns this list
)
)
procedure( uwlMakePin( cv netName figure direction)
  let( ( net trm pin)

    net = dbCreateNet( cv netName)
    trm = dbCreateTerm( net netName "inputOutput")
    pin = dbCreatePin( net figure netName)
    pin->accessDir = direction
    pin ; return the pin object id
  )
)
)
```

SKILL for Thin-Film Resistor Pcell

```
pcDefinePCell(
; library, cellname and view:
  list( ddGetObj( "pSamples") "indCirc" "layout")

; input parameters and their default values:
  ((di_pc      100.)
   (do_pc      400.)
   (width_pc   20.)
   (spacing_pc 15.)
   (layerNum   1)
  )
  let( (pointLists x wh fig y)

; build the spiral shape:
  pointLists = uwlSpiralPaths( di_pc do_pc width_pc spacing_pc 72)
  foreach( path pointLists
    dbCreatePath( pcCellView layerNum path width_pc)
  )

; create the nets, terminals and pins:
  x = do_pc * .5
  wh = width_pc * .5
  fig = dbCreateRect( pcCellView layerNum
    list( x:-wh x+width_pc+spacing_pc:wh))
  uwlMakePin( pcCellView "n1" fig '("right"))

  x = round( caaar( pointLists))
  y = round( cadaar( pointLists))
  fig = dbCreateRect( pcCellView layerNum
    list( x-wh:y-wh x+wh:y+wh))
  uwlMakePin( pcCellView "n2" fig '("right" "top" "left" "bottom"))
  dbReplaceProp( pcCellView "instNamePrefix" "string" "ind")
)
```

Virtuoso Parameterized Cell Reference

Creating Graphical Parameterized Cells

)) t

Debugging SKILL Pcells

The *Pcell IDE* application is available in the *Launch* menu of VLS, VSE, and VSE-Symbol editor windows. The Pcell IDE assistant is displayed to the left of the design display area in the layout or schematic window. You can use the Pcell IDE window to view Pcell attributes and connectivity information while the Pcell is being debugged.

Using Pcell IDE you can debug Pcell supermaster, instances, and abutment.

Working with Pcell IDE

Debugging a Pcell is an iterative process. So, you can specify different parameters or select different Pcell supermaster at the end of the Pcell evaluation to start a new iteration. A Pcell is composed by multiple SKILL functions contained in multiple SKILL files. The file that contains the `pcDefinePCell` function call is the Pcell definition file. All other files are the Pcell support files.

To debug a Pcell using Pcell IDE requires the Pcell Support files to be loaded. Typically, the Pcell Support files are loaded through the `libInit.il` file. It is recommended to load the Pcell Definition file from the Pcell IDE assistant. You can also load this file from the SKILL IDE window but this is no longer the recommended method.

After the Pcell Definition file is loaded, you can specify the appropriate Pcell parameters to be evaluated and the necessary breakpoints in the IDE source window so that the intermediate Pcell evaluation result can be examined. You can also select the graphical shape in the window that displays the Pcell supermaster or submaster contents to lookup the source code that generated the selected shape.

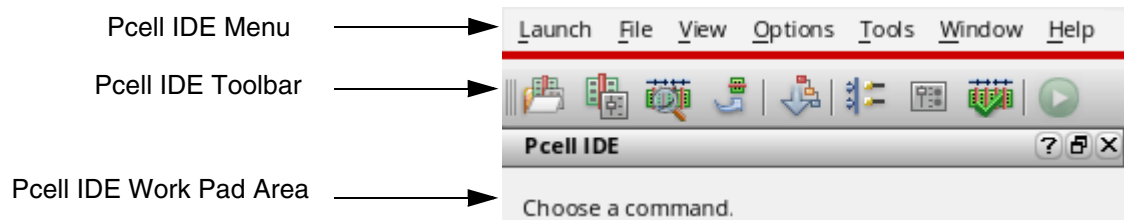
Open the Pcell IDE Window

To open Pcell IDE, select *Launch – Pcell IDE* from any VLS, VSE, or VSE Symbol editor windows.

Virtuoso Parameterized Cell Reference

Debugging SKILL Pcells

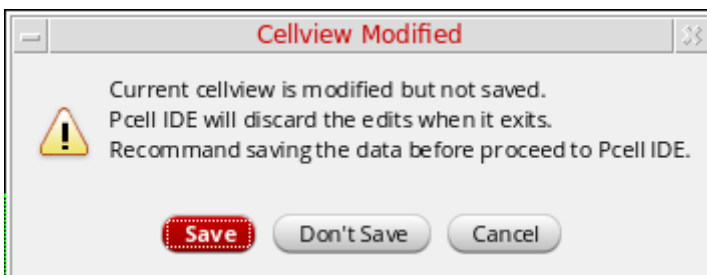
Note: Pcell IDE has its own menu bar, toolbar, bindkeys, and work pad area. The Pcell IDE menus are based on their root application menu with all editable commands are removed. This is because Pcell IDE does not allow you to edit the Pcell data during the debugging process.



Important

If you modify the cellview and launch Pcell IDE window without saving the cellview, the Cellview Modified dialog box is displayed. You can choose any of the following options:

- ☐ **Save:** It saves the current cellview and launches the Pcell IDE assistant.
- ☐ **Don't Save:** It does not save the current cellview and launches the Pcell IDE assistant. You might lost the changes made to the cellview while exiting Pcell IDE.
- ☐ **Cancel:** It cancels the launch of the Pcell IDE assistant.



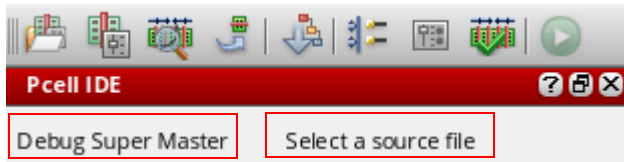
The Work Pad area of the Pcell IDE window displays the following information:

- The text in the top left corner shows the current active command.

Virtuoso Parameterized Cell Reference

Debugging SKILL Pcells

- The text in the top right corner shows either the current status or prompt of IDE.



Only one Pcell IDE session is supported at a time. If you launch Pcell IDE while one is already active in another window, the following dialog box is displayed.



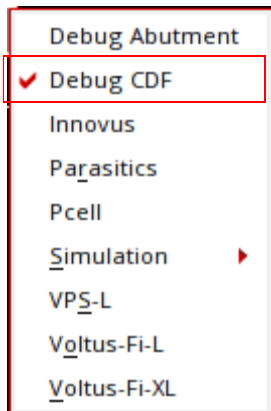
Note: Upon launching the Pcell IDE assistant, the menu items *Save*, *Save a Copy*, *Discard Edits*, and *Save Hierarchically* will be disabled from the *File* menu of VLS, VSE, and VSE Symbol editor windows.

Automatically Install Debug CDF Plugin

The Debug CDF assistant is opened automatically by the Pcell IDE through the default Pcell IDE workspace. The Debug CDF assistant is placed on the second tab of the Pcell IDE assistant window. Once you launch the Pcell IDE window, the *Launch — Plugins — Debug CDF* option is checked.

Virtuoso Parameterized Cell Reference

Debugging SKILL Pcells




However, once you exit Pcell IDE, the *Launch — Plugins — Debug CDF* option will be unchecked.

Pcell IDE Window Toolbar

Commands on the Pcell IDE window toolbar are grouped as follows, based on the functionality the commands provide:

- Debug
 - ❑ [Super Master Command](#)
 - ❑ [Instance Command](#)
 - ❑ [Debug Abutment Command](#)
- Tools
 - ❑ [Publish Command](#)
 - ❑ [Show Connectivity Command](#)
 - ❑ [Check Parameter Command](#)
 - ❑ [Debug Abutment Command](#)
 - ❑ [Run Command](#)

In addition, the *Run* command is also available on the toolbar.

Note: Clicking the hide  button in the upper-right corner of the Pcell IDE assistant only closes the window. However, the Pcell IDE session will continue. To exit pcell IDE session you need to launch another application using the *Launch* menu or close the window where Pcell

IDE was launched from.

Let's consider a scenario where you launch Pcell IDE from the `xyz` window. Now, after you execute the various debug commands, additional windows can be opened by Pcell IDE. However, if you close those additional windows, the Pcell IDE does not terminate.

Now, to terminate Pcell IDE you either need to launch another application from the `xyz` window or close the `xyz` window. As Pcell IDE exits, if any additional windows are opened then they will be closed by Pcell IDE as a part of the exit clean up process.

Also, notice that the Launch menu entries from those additional windows will be disabled.

Description

The following is a brief description of the available commands.

Super Master Command

You use the *Debug Super Master* command to step through the evaluation of the Pcell supermaster. Using this command an empty (read-only) cellview gets created when you specify the source file with the `pcDefinePcell` function. Once the evaluation is completed, the shapes according to the Pcell default values are drawn in the cellview. This command enables you to resolve the Pcell source code errors.

Instance Command

You use *Debug Instance* command to directly debug a selected instance's submaster without leaving the design area. After a proper instance is selected, the parameters are updated according to the selected instance's parameters values.

Debug Abutment Command

You use the *Debug Abutment* command to simulate the Pcell abutment without using the connectivity engine of VLS XL and execute the user-defined abutment functions.

Publish Command

You use the *Publish* command to update or redefine original Pcell supermaster from Pcell IDE after the debug operation is completed.

Debug Hier Mode Command

You use the *Debug Hier Mode* command enables you to debug supermaster or instances at all displayed hierarchy levels.

Show Connectivity Command

You use the *Show Connectivity* command to view the Pcell instance's connectivity model. This command displays the status of each terminal as weak connect, strong connect, or must connect. In addition, you use this command to view the information and properties for each terminal and pin figures.

Check Parameter Command

You can use the *Check Parameter* command to display the selected Pcell instance's parameter name, type, default value, and instance value.

Check Abutment Command

You can use the *Check Abutment* command to analyze the abutment information that is stored in the database.

Run Command

You use the *Run* command to start debugging a Pcell. After you setup the parameter values, the *Run* icon is enabled. You can start another evaluation by selecting the *Run* icon again at the end of the evaluation process. This *Run* icon is disabled in any of the following conditions:

- During Pcell evaluation.
- If supermaster source code is not loaded.
- If you have selected any of the non-debugging commands.

Virtuoso Parameterized Cell Reference

Debugging SKILL Pcells

Debugging Pcell Supermaster

To debug Pcell supermaster, you need to perform the following steps:

1. Click *Debug Super Master* icon from the toolbar

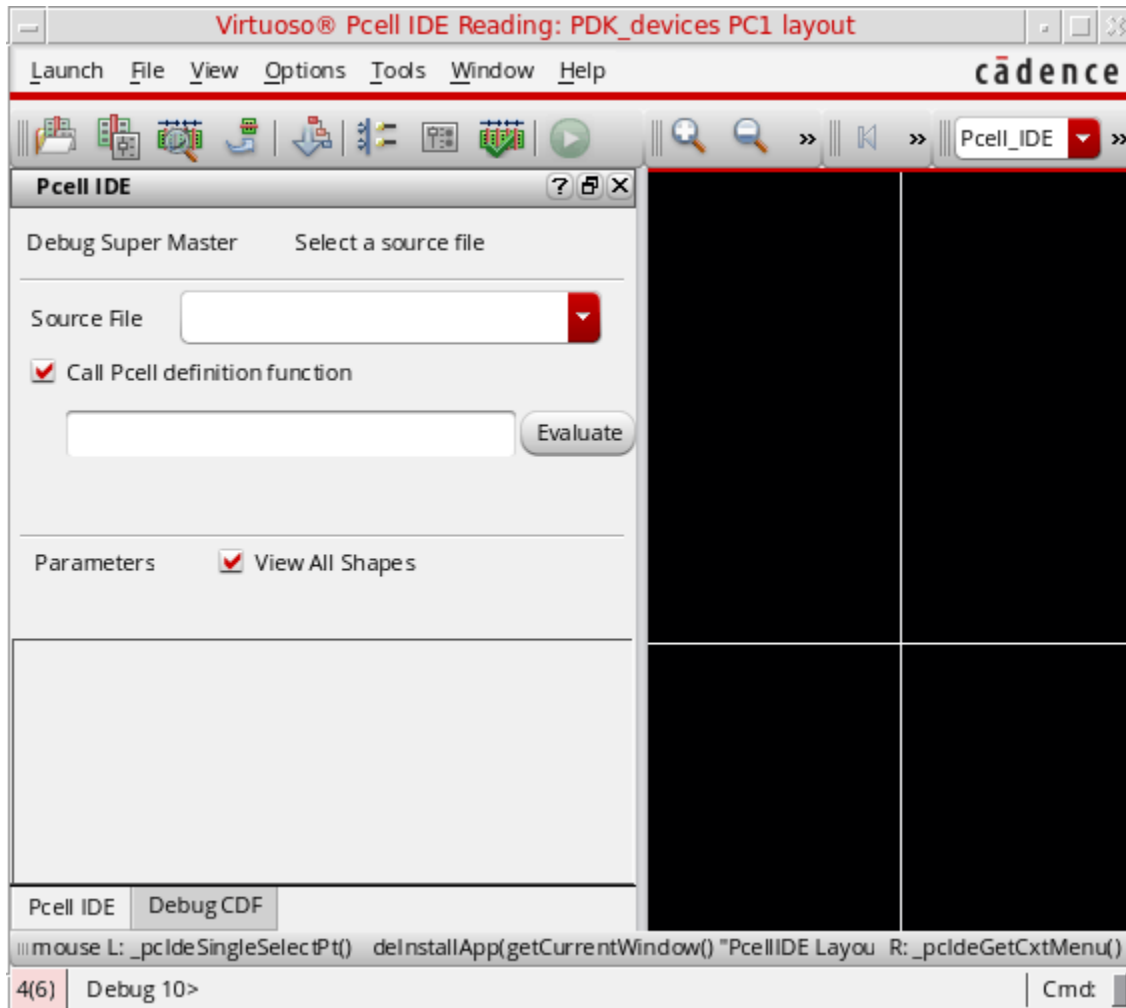


2. The fields related to debug supermaster are displayed in the Work Pad area. Once you specify the source file with the `pcDefinePcell` function, a new layout named `<lib>
<supermasterCellName> "layout_DBG"` is opened. This cellview is deleted once

Virtuoso Parameterized Cell Reference

Debugging SKILL Pcells

the Pcell IDE session is terminated.



3. From the *Source File* drop-down menu, select the Pcell definition file you need to debug.
4. Select the *Call Pcell definition function* check box and add the procedure name in the text box.

Note: You can select the *Call Pcell definition function* check box only when the `pcDefinePcell` or `dbDefineProc` function is called from inside the procedure.

5. Click the *Evaluate* button. It creates a temporary cellview, `<lib> <pcellName> "layout_DBG"`.
6. The *Run* icon gets enabled on the Pcell IDE toolbar. Click the *Run* icon to start the evaluation process.

Virtuoso Parameterized Cell Reference

Debugging SKILL Pcells

7. Once you click the *Run* icon, Pcell IDE sets an initial breakpoint at the first line of the source code body before starting the Pcell evaluation.

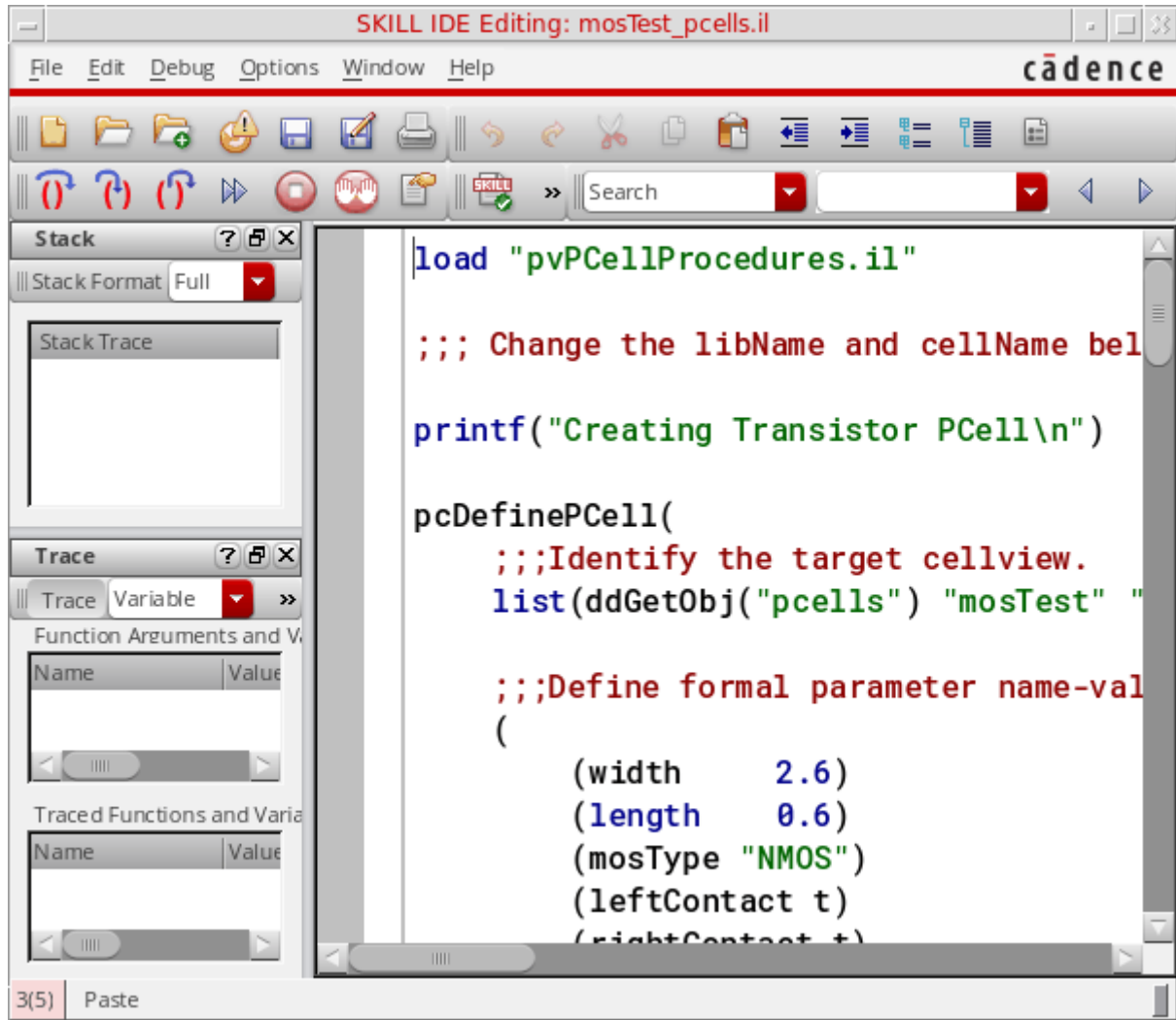
Note: If `pcDefinePcell` is specified inside a procedure, then you would have to call that procedure in CIW and click the *Evaluate* button to start the debugging process. Otherwise, the debugging process will start automatically.

Note: If there are multiple `pcDefinePcell` specified inside a procedure, then multiple empty cellview windows will be displayed. Each empty cellview window is associated to its corresponding debugged Pcell supermaster. You can select the desired cellview tab window that needs to be debugged and then click the Pcell IDE's *Run* icon to start debugging the selected Pcell supermaster.

8. The SKILL code corresponding to the supermaster is displayed in the SKILL IDE window. Set breakpoints and walk through the code using the *Next*, *Step*, and *Continue* icons.

Virtuoso Parameterized Cell Reference

Debugging SKILL Pcells

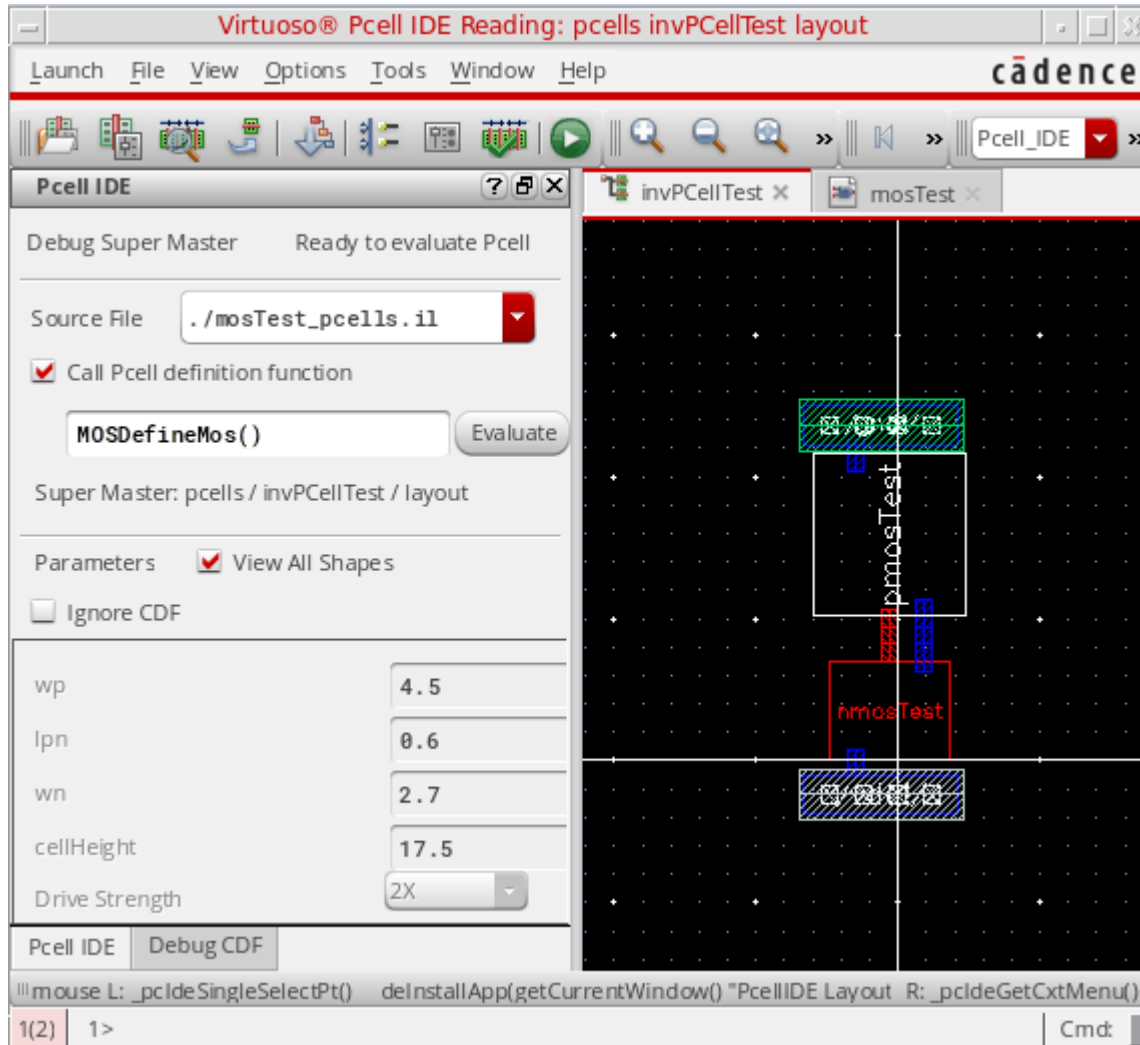


9. On the SKILL IDE window, click the *Step* icon to step through the Pcell source code. The geometry is incrementally updated and the Pcell supermaster is displayed in the layout

Virtuoso Parameterized Cell Reference

Debugging SKILL Pcells

window.



10. After the evaluation is successfully completed, the Pcell default parameters are displayed in the Pcell IDE assistant. However, these parameters are not editable.

Note: You can restart the debugging process once the *Run* icon is enabled.

After Pcell evaluation you can select shapes in the graphic window to view the corresponding code in the SKILL IDE window.

Virtuoso Parameterized Cell Reference

Debugging SKILL Pcells

Important

The *View All Shapes* check box is selected by default. If this check box is selected, then at the end of Pcell evaluation, the shapes will be displayed in the Zoom To Fit view in the graphic window.

Note: The *Display Stop Level*



text box displays the current window's stop level value.

Virtuoso Parameterized Cell Reference

Debugging SKILL Pcells

Debugging CDF

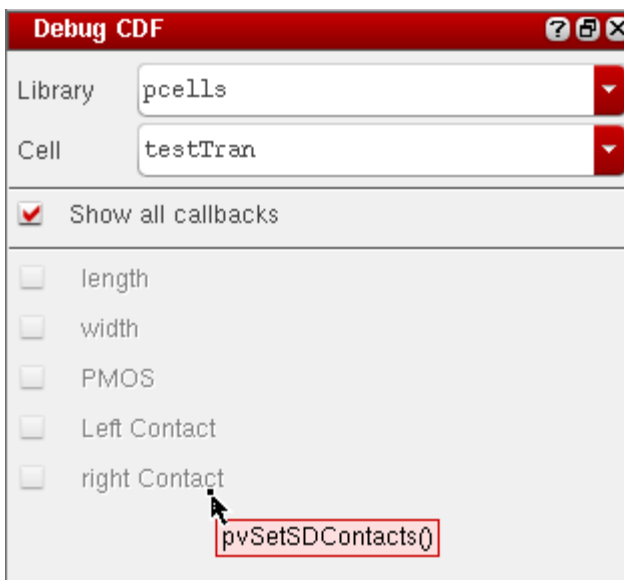
Debug CDF is a plugin that is used to debug CDF callback procedures using simple SKILL expressions for form initialization (`forminit`), form completion (`doneProc`), and any parameter field value modification (parameter callbacks).

The Debug CDF plugin:

- Is available in Pcell IDE, Schematic, and Layout applications.
- Is installed automatically by Pcell IDE through the default Pcell IDE workspaces.
- Creates the Debug CDF assistant that allows you to select CDF callbacks for debugging.

For more information on debugging CDF, refer to the [Debugging Callbacks](#) section of the *Component Description Format User Guide*.

You can also view the parameter definition as a tooltip by pointing on a parameter name in the Debug CDF assistant, as shown below.



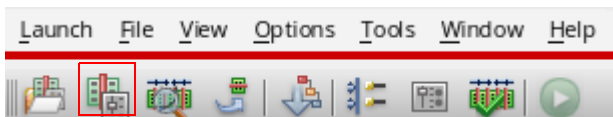
Virtuoso Parameterized Cell Reference

Debugging SKILL Pcells

Debugging Pcell Instances

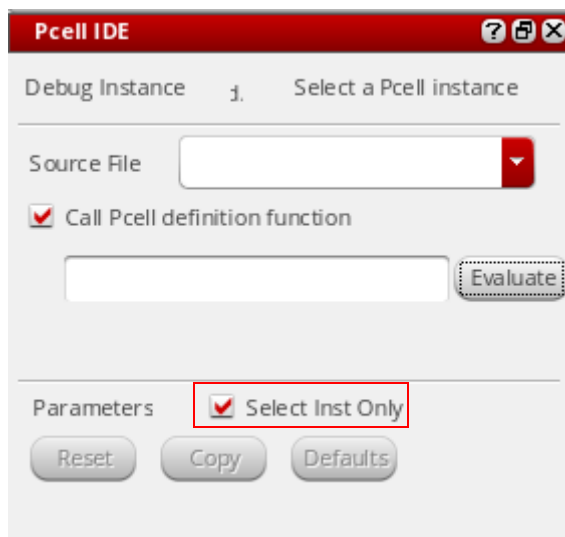
To debug Pcell instances, you need to perform the following steps:

1. Click *Debug Instance* icon from the Pcell IDE toolbar.



Important

In the Debug Instance form, the *Select Inst Only* check box is selected by default. When it is set, you can only select instances.



This feature allows you to select the desired instance for debugging. You can debug an instance from any of the displayed hierarchical levels. However, you must select only one instance at a time for debugging.

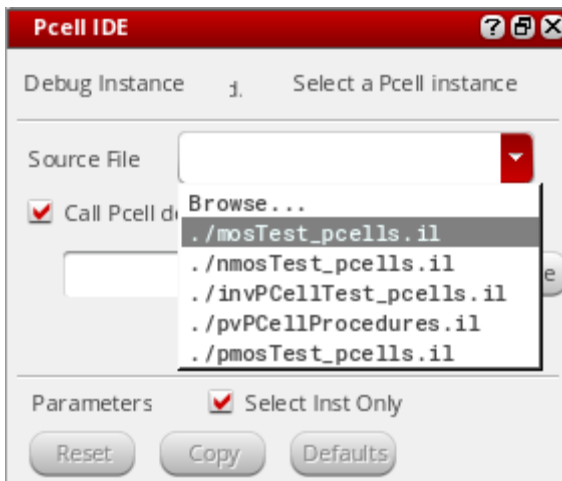
Before you run the *Debug Instance* command, you can select an instance or any

Virtuoso Parameterized Cell Reference

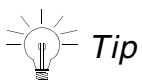
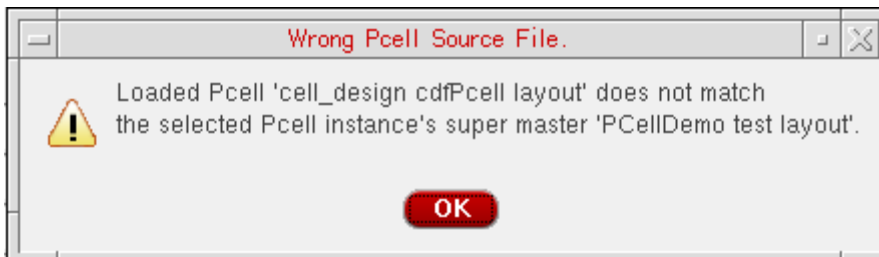
Debugging SKILL Pcells

object. If the preselected object is a Pcell instance, its parameters will be displayed accordingly.

2. From the *Source File* drop-down menu, select the Pcell definition file you need to debug.



Note: If the Pcell source of the selected instance is not loaded, then you cannot start the debugging process. In addition, the error message will be displayed (as shown in the figure below) if the source file does not match with the supermaster.



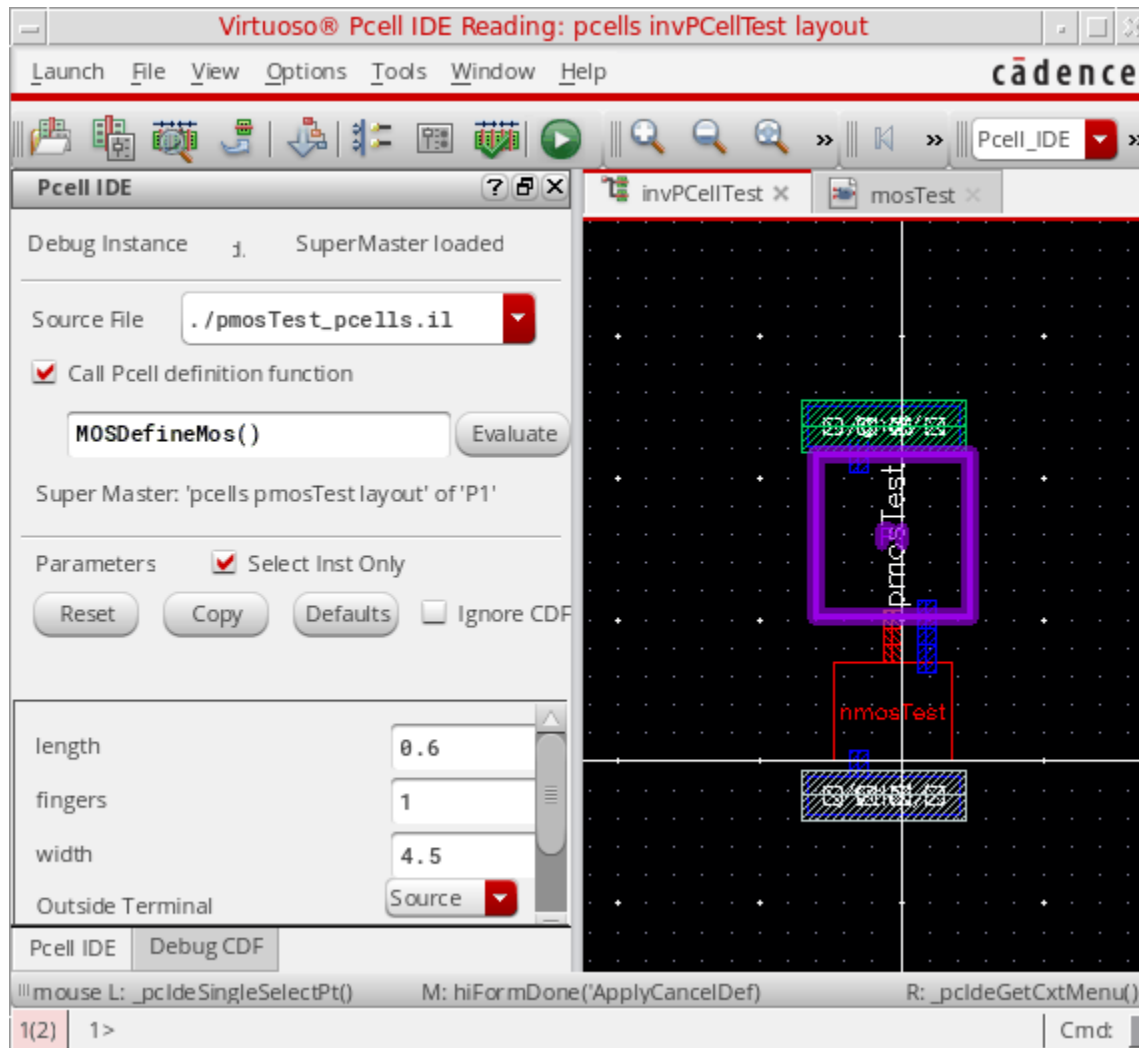
Tip

In case you specified the incorrect source file in the Pcell IDE window, you can load the correct version of the source file directly from the SKILL IDE window.

Virtuoso Parameterized Cell Reference

Debugging SKILL Pcells

3. Next, you need to select an instance to load the supermaster.



4. Select the *Call Pcell definition function* check box and add the procedure name in the text box.

Note: You can select the *Call Pcell definition function* check box only when the `pcDefinePcell` or `dbDefineProc` function is called from inside the procedure.

5. Click the *Evaluate* button. This allows Pcell IDE to confirm if the Pcell source file and the selected Pcell instance are matching.

After an instance is selected and its Pcell source code is loaded, the *Run* icon gets enabled. Once you click the *Run* icon, the *Select Inst Only* check box is deselected, which allows you to select any object during Pcell evaluation for cross highlight purpose.

Virtuoso Parameterized Cell Reference

Debugging SKILL Pcells

However, after a Pcell evaluation is complete, the *Select Inst Only* check box is selected again.

You can deselect the *Select Inst Only* check box anytime to perform cross highlight between object and source code.

Note: You can choose to ignore the CDF values using the *Ignore CDF* check box or modify any of the parameter values.

Important

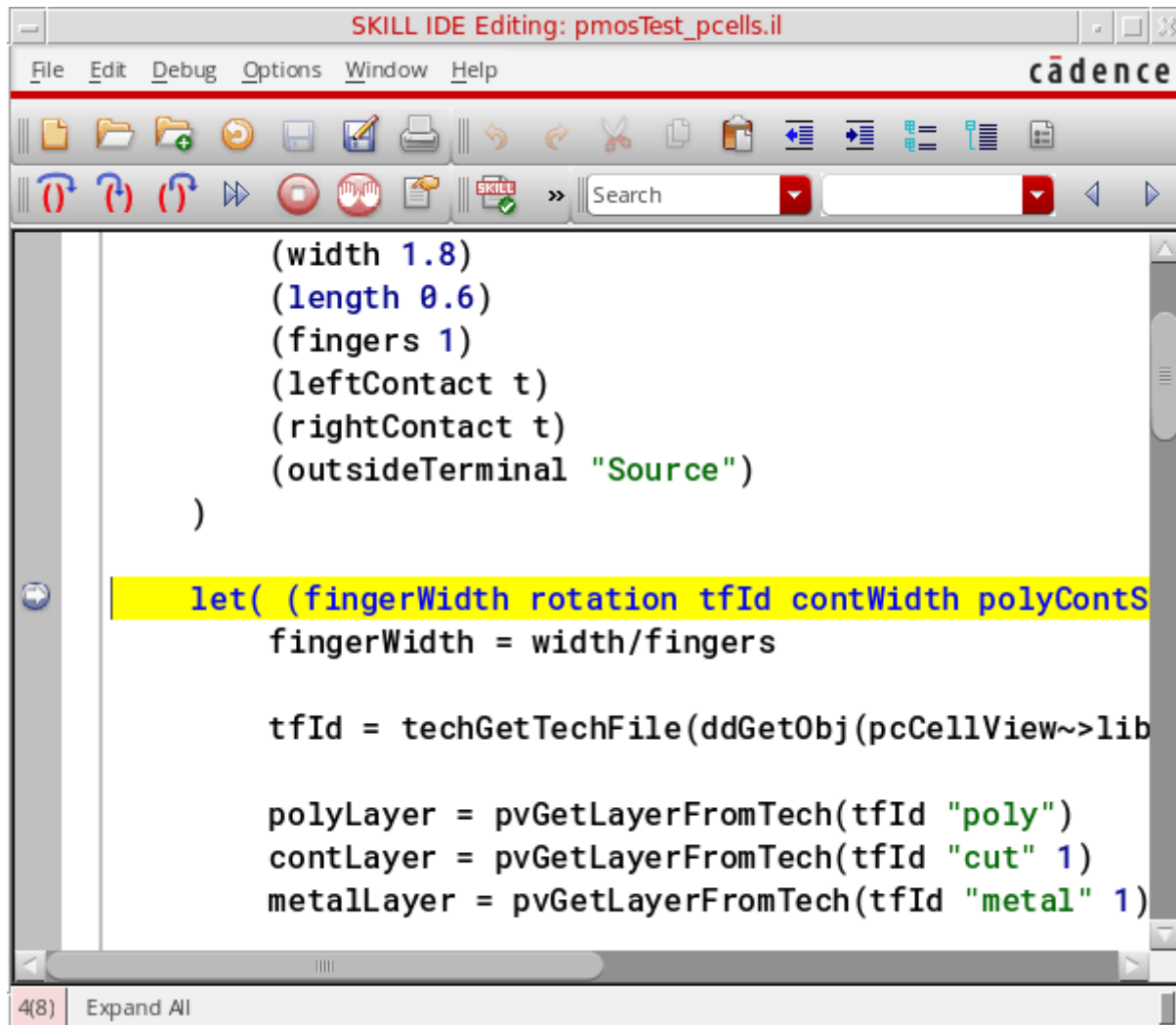
You can click the *Reset* button to reset the parameter values to the last instance parameter values. If the current instance had been re-evaluated, Pcell IDE remembers the new set of parameter values. Therefore, after modifying the parameter values, you can always restore the original instance's settings.

However, this is different from the *Defaults* button, which sets all the parameter values to their default values.

6. The SKILL code corresponding to the supermaster is displayed in the SKILL IDE window. Set breakpoints and walk through the code using the *Next*, *Step*, and *Continue* icons.

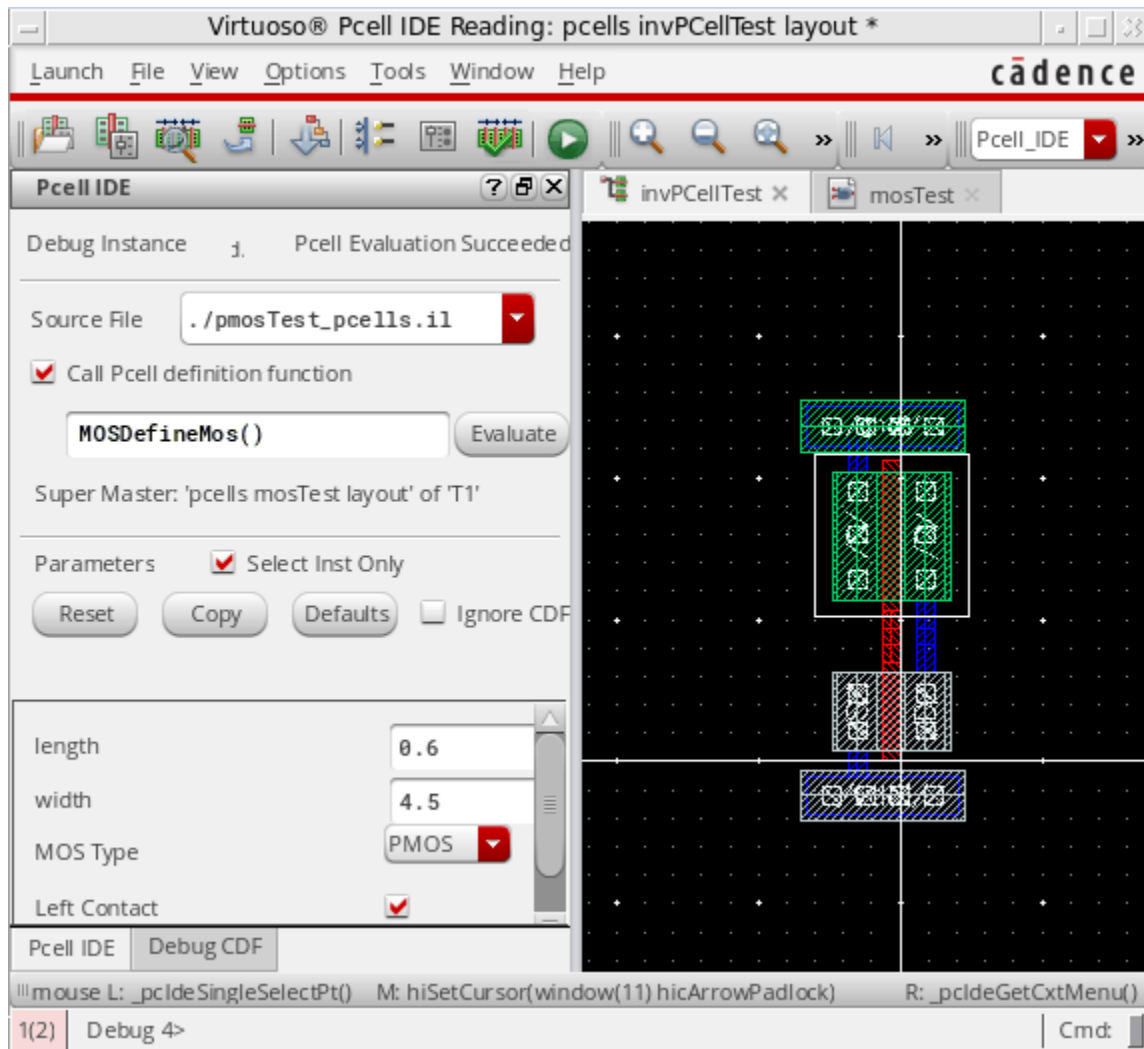
Virtuoso Parameterized Cell Reference

Debugging SKILL Pcells



7. The Run icon gets enabled on the Pcell IDE toolbar. Click the *Run* icon to start the evaluation process.
8. On the SKILL IDE window, click the *Step* icon to step through the Pcell source code. The graphic is incrementally updated and the Pcell submaster is displayed in the layout window.

Virtuoso Parameterized Cell Reference Debugging SKILL Pcells



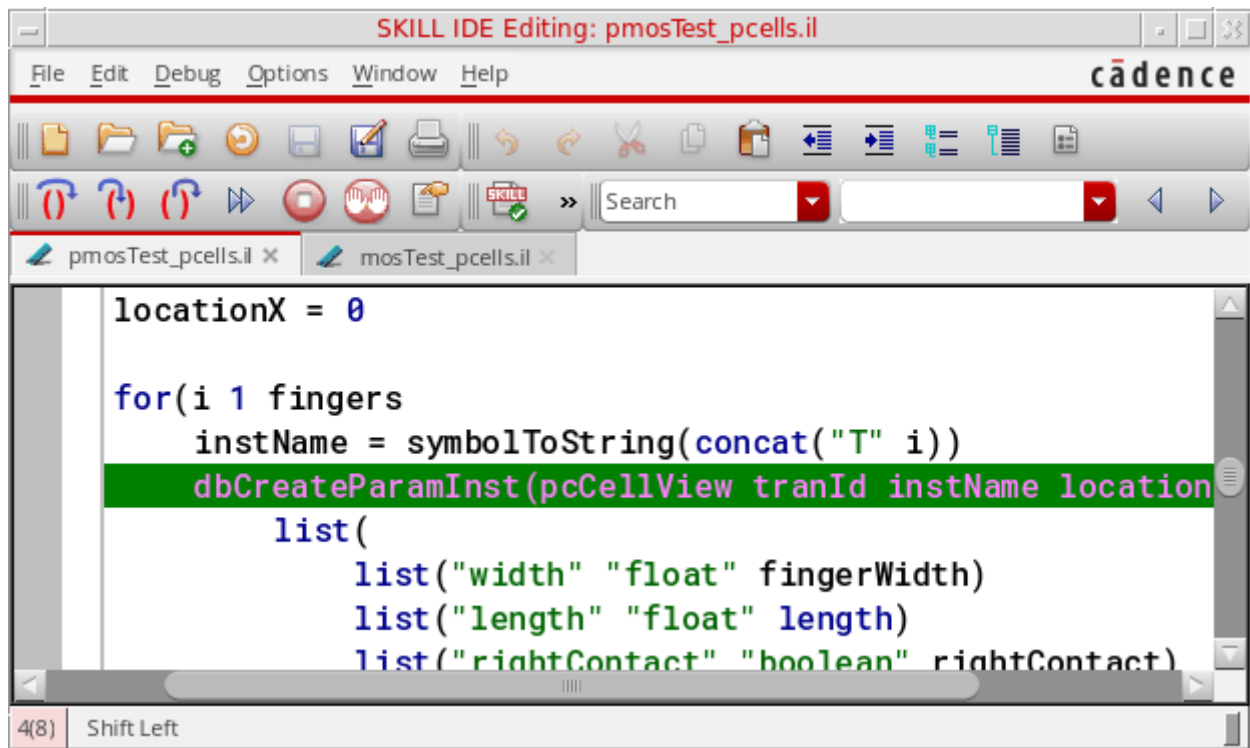
9. After Pcell evaluation you can select shapes in the graphic window to view the corresponding code in the SKILL IDE window.

Important

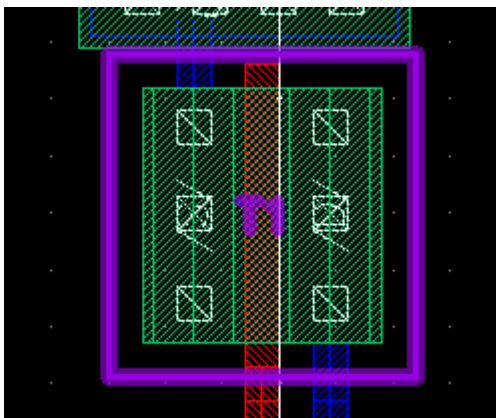
Alternatively, if you select a single line code in the SKILL IDE window, the corresponding shape will be highlighted in the graphic window, as shown below.

Virtuoso Parameterized Cell Reference

Debugging SKILL Pcells



Text Selected in the SKILL IDE Window



Shape Selected in the Graphical Window

In addition, if a database shape object is referenced by multiple source lines, then you can use the *Cross Highlight* toolbar to visit all the source lines where this object was created and modified. To view the *Cross Highlight* toolbar, you need to right-click the toolbar area and then select the *Cross Highlight* option from the context-sensitive menu, as shown below.

Virtuoso Parameterized Cell Reference

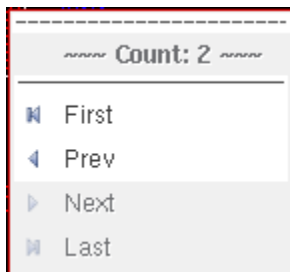
Debugging SKILL Pcells



The *Cross Highlight* toolbar will be displayed, as shown below.




Alternatively, to view the source code of an object, which is scattered at multiple locations, you can right-click the object and select appropriate options from the context-sensitive menu, as shown below.



Publish Command

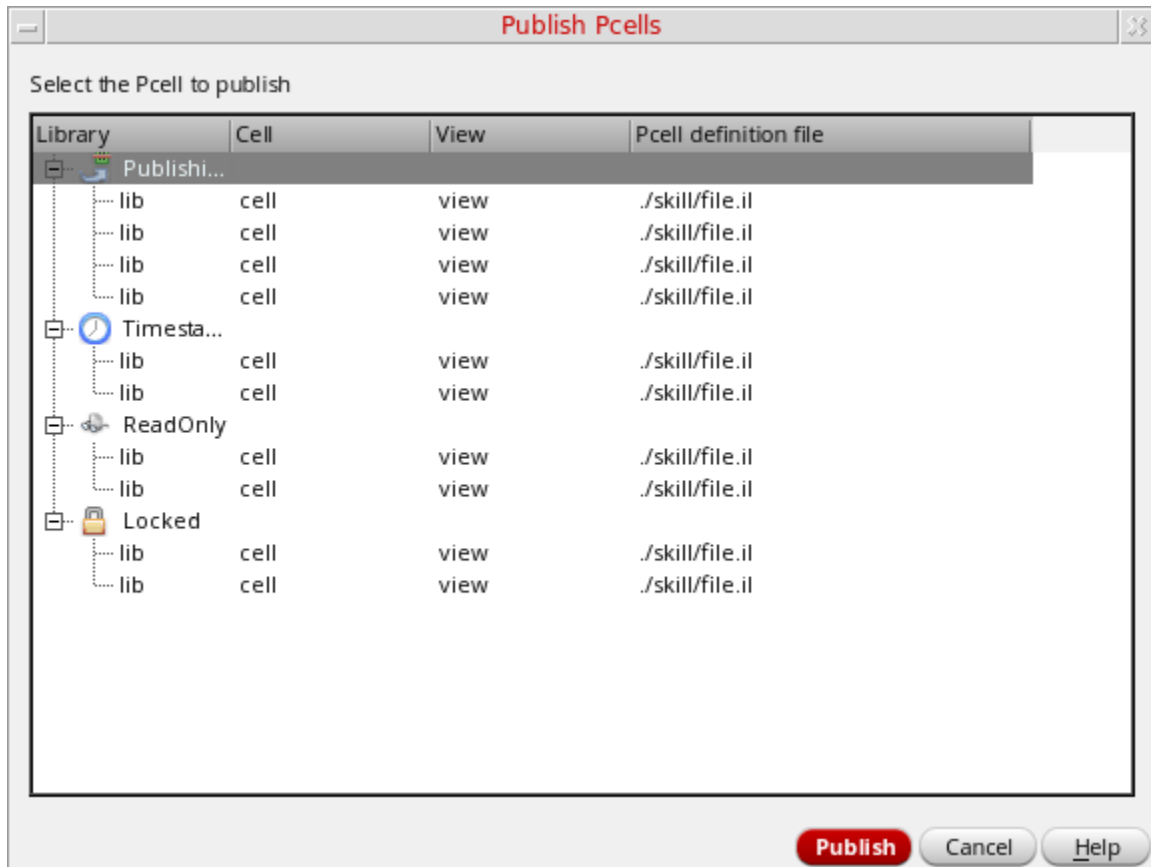
The *Publish* command allows you to update the original Pcell supermaster view directly from Pcell IDE after the debug operation is complete. Earlier, to fix a bug or to enhance some functionality, you had to modify the Pcell source file during the debug operation. To do this, you were required to save the Pcell source file, and then load it in CIW to regenerate the Pcell supermaster.

Now you can update or redefine the original Pcell supermaster from Pcell IDE by using the *Publish* command. The *Publish*  button is disabled by default on the Pcell IDE toolbar,

Virtuoso Parameterized Cell Reference

Debugging SKILL Pcells

and it gets enabled after the first Pcell debug session. After you click the *Publish* button, the Publish Pcells dialog box is displayed, as shown below.



This dialog box lists the following categories (expandable) and buttons:

Publishing – Lists Pcells that are ready to be published.

Timestamp – Lists Pcells that have timestamp violation. Their original supermaster cellviews are newer than the debugged ones. They are published after user confirmation.

ReadOnly – Lists the original Pcell supermaster cellviews that are not writable, and therefore cannot be published.

Locked – Lists the original Pcell supermaster cellviews that are locked, and therefore cannot be published.

Publish button – Regenerates the selected supermasters.

Cancel button – Cancels the *Publish* command.

Virtuoso Parameterized Cell Reference


Debugging SKILL Pcells

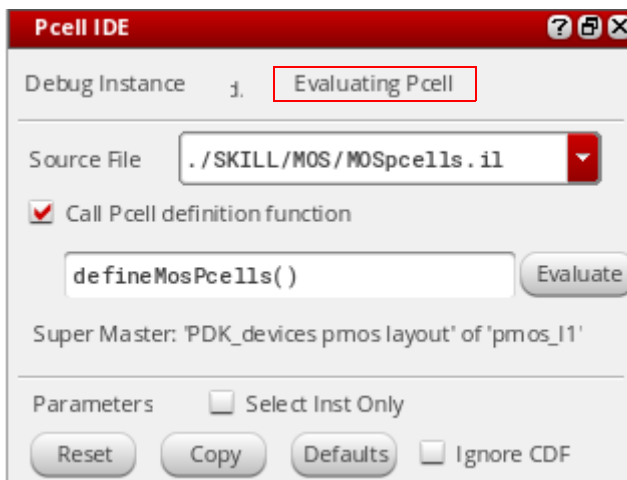
Help button – Opens the help documentation in this area.

Note: This functionality is available only when the supermaster is created using the `pcDefinePCell()` or `pcGenCell()` function.

Debug Hier Mode Command

You can also debug supermaster or instances at all displayed hierarchy levels. To do this:

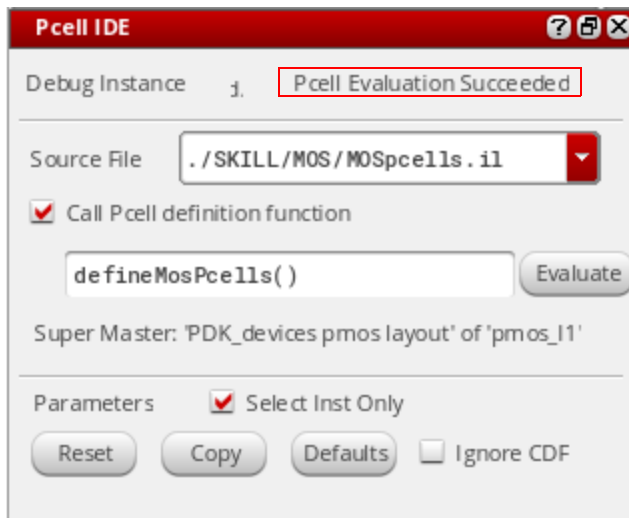
1. You need to click the *Debug Hier Mode*  icon from the Pcell IDE assistant toolbar.
2. Now, once you click the *Continue* icon on the SKILL IDE window.
3. On the Pcell IDE assistant, the Evaluating Pcell message is displayed.



4. You again need to click the *Continue* icon from the SKILL IDE toolbar to complete the Pcell evaluation process.

Virtuoso Parameterized Cell Reference

Debugging SKILL Pcells



Video

To view the demonstration on debugging hierarchical Pcells, see [Debugging Hierarchical Pcells](#) video.

Note: Access to this video will depend on the availability of a web browser and a Cadence Online Support account.

Pcell Evaluation Message

When you debug a Pcell design using either the Debug Super Master or Debug Instance command, the following Pcell evaluation message will be displayed in CIW:

Debug Super Master

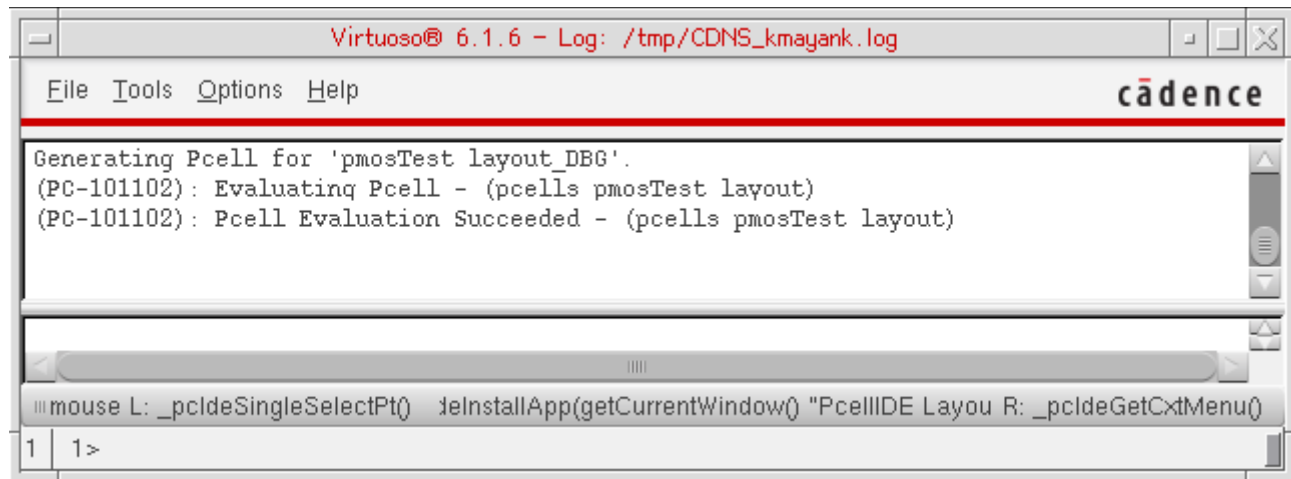
```
Evaluating Pcell - (libName cellName viewName)
Pcell Evaluation Succeeded - (libName cellName viewName)
Pcell Evaluation Failed - (libName cellName viewName)
```

Debug Instance command

```
Evaluating Pcell - Instance | P0 (libName cellName viewName)
Pcell Evaluation Succeeded - Instance | P0 (libName cellName viewName)
Pcell Evaluation Failed - Instance | P0 (libName cellName viewName)
```

Virtuoso Parameterized Cell Reference

Debugging SKILL Pcells



Environment Variable

`cdba dbLogPcellWarnings` boolean `t`

This environment variable allows you to redirect warning messages, which are generated during the Pcell evaluation, in the `/tmp` directory.

By default, only the error messages are redirected to the pcell error log file. However, when this environment variable is set to `true`, warning messages are also redirected to the pcell error log file.

Supported and Not Supported Pcell Instances

A list of supported and not supported Pcell instances is illustrated in the table below:

Supported Pcell Instances	Not Supported Pcells Instances
SKILL Pcells	Non-SKILL Pcells
SKILL++ Pcells	Expressed Pcells
Mosaic Pcells	

Virtuoso Parameterized Cell Reference

Debugging SKILL Pcells

Important

The mosaic Pcell instance is supported but only the first item will be highlighted to reflect the evaluation progress.

Viewing Pcell Connectivity Model

The *Show Connectivity* command allows you to view the Pcell instance's connectivity model. This command is supported only for the top level design.

The Work Pad area for this command consists of the list of current editing cellview's of Pcell instances in a tree structure. Each instance's terminals are listed under the instance level and each terminal pins are listed under each terminal level.

Note: If you right-click anywhere inside the Pin Connectivity area, a context-sensitive menu is displayed that has four options: *Expand*, *Collapse*, *Expand All*, and *Collapse All*.

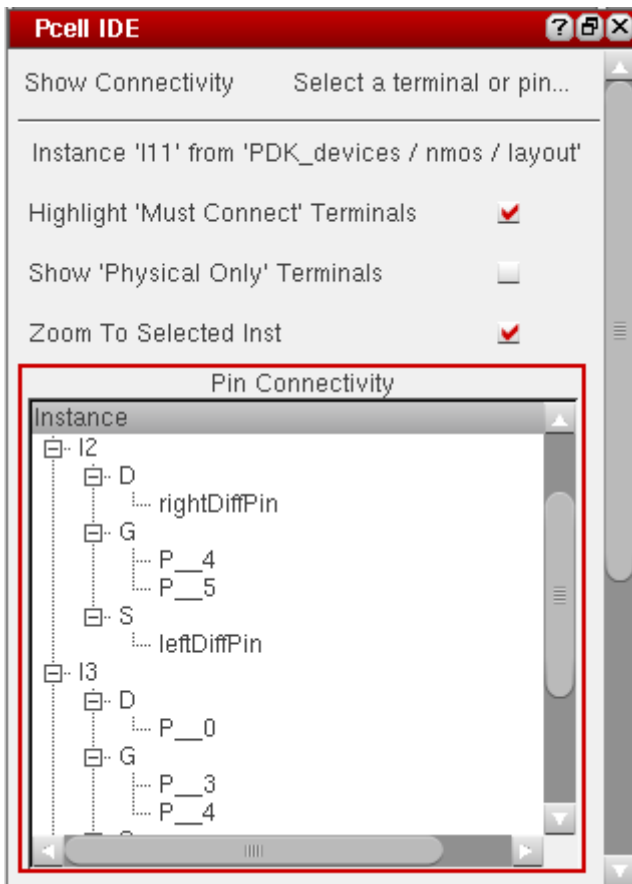
The *Expand* option expands the entire branch either at the instance or terminal level and the *Collapse* option collapses the branch. These two menu items are disabled at the pin branch level, because pin branch is the lowest branch and cannot be expanded or collapsed.

The *Expand All* option expands all nodes in the tree and the *Collapse All* option collapses all nodes of the tree.

Note: The tree is sorted alphabetically based on the instance name.

Virtuoso Parameterized Cell Reference

Debugging SKILL Pcells

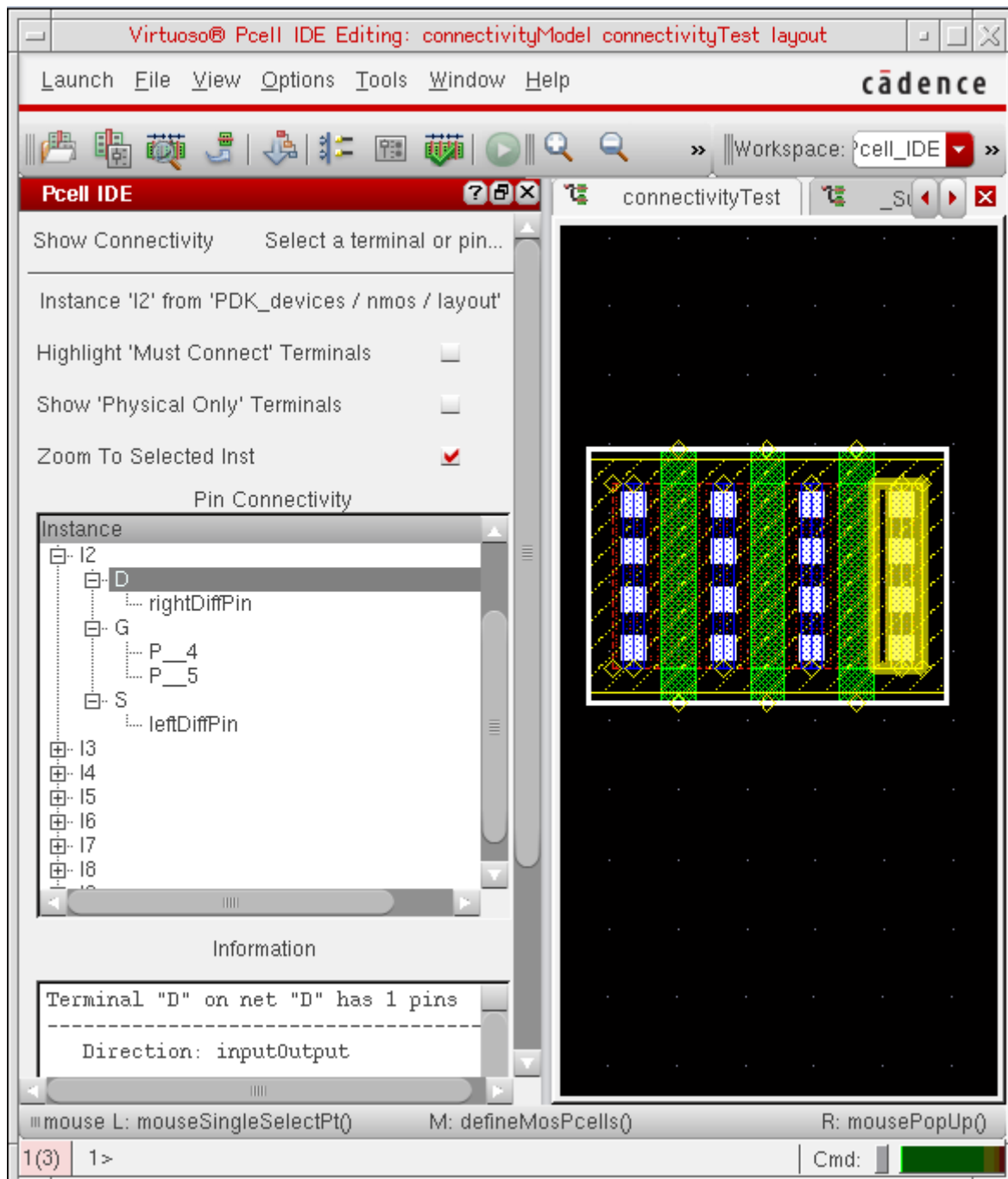


Only top level instances are selectable in the graphic window for this command. When you select an instance node from the instance tree, the corresponding instance in the graphic window is highlighted in red and instance name is displayed. In addition, if you select an instance from the graphic window, the corresponding instance node on the instance tree from the Word Pad area is selected.

The Work Pad area of this command consists of the following options:

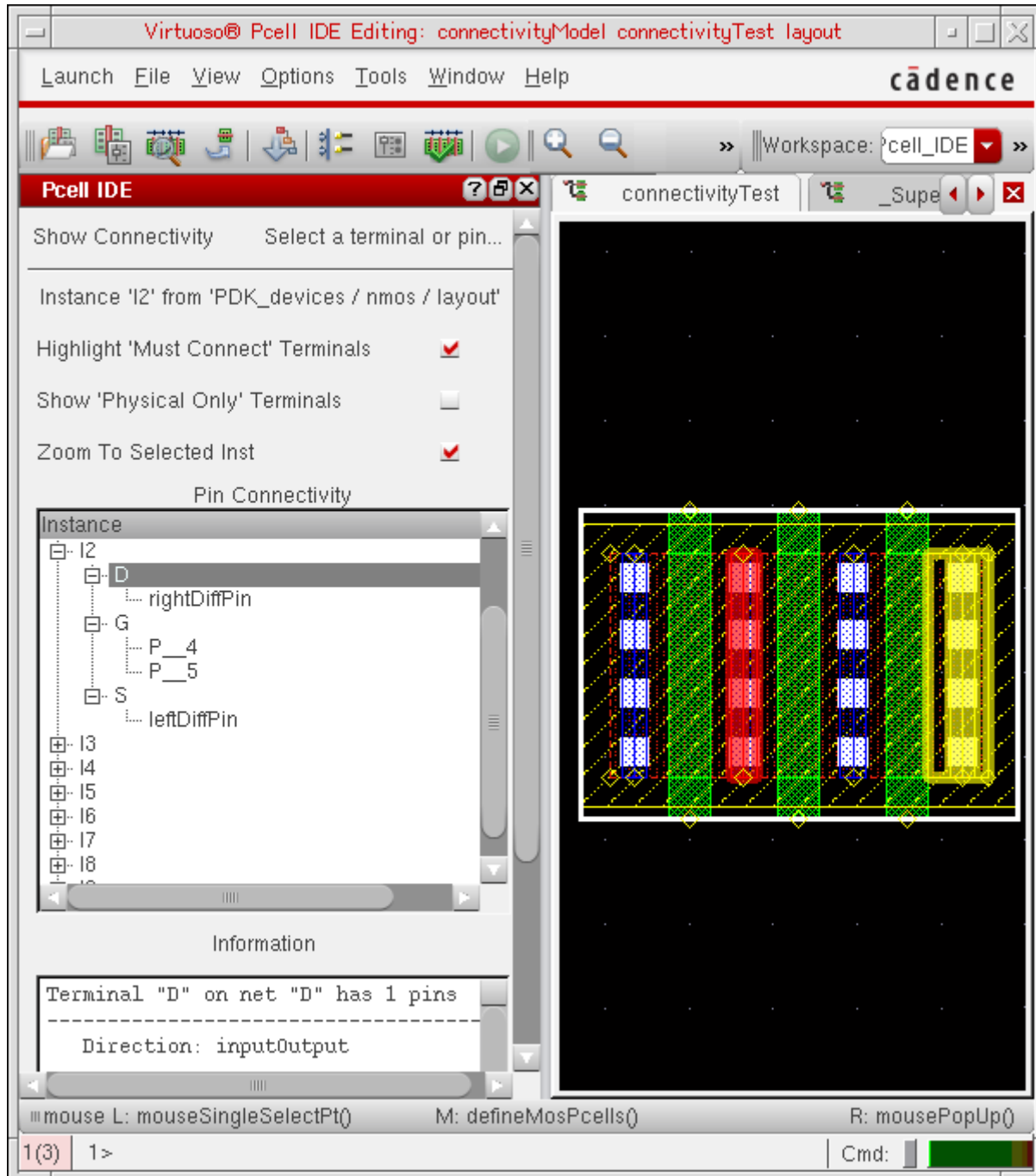
- **Highlight 'Must Connect' Terminals:** You can use this option to highlight the “must connect” terminals, if applicable, of the selected terminal. If the *Highlight 'Must Connect' Terminals* check box is not selected, then only the selected terminal will be highlighted in yellow, as shown in the figure below.

Virtuoso Parameterized Cell Reference Debugging SKILL Pcells



However, If you select the *Highlight 'Must Connect' Terminals* check box, then all must connect terminals of the selected terminal gets highlighted in different colors, as shown in the figure below.

Virtuoso Parameterized Cell Reference Debugging SKILL Pcells

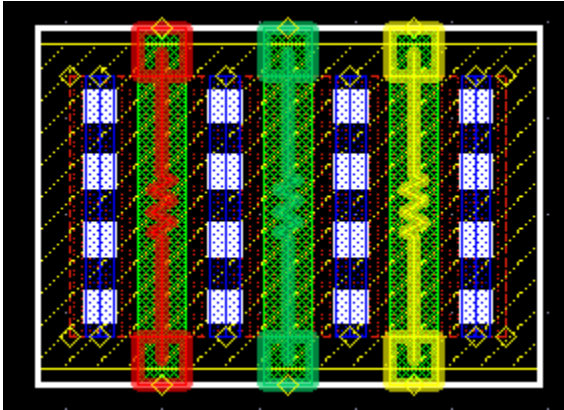


In addition, when the terminal is a “weak connect”, a resistor is drawn between the pins. However, If it is a “strong connect”, then a straight line is drawn between the pins, as

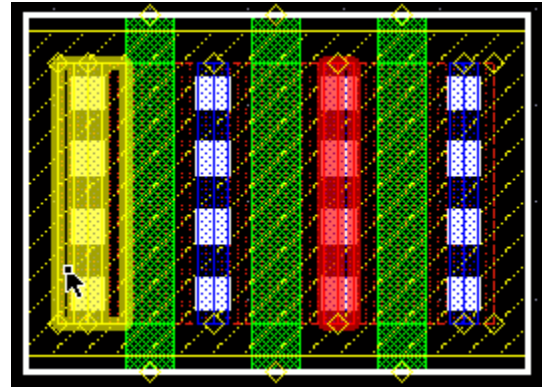
Virtuoso Parameterized Cell Reference

Debugging SKILL Pcells

shown in the figure below.



Weak Connect

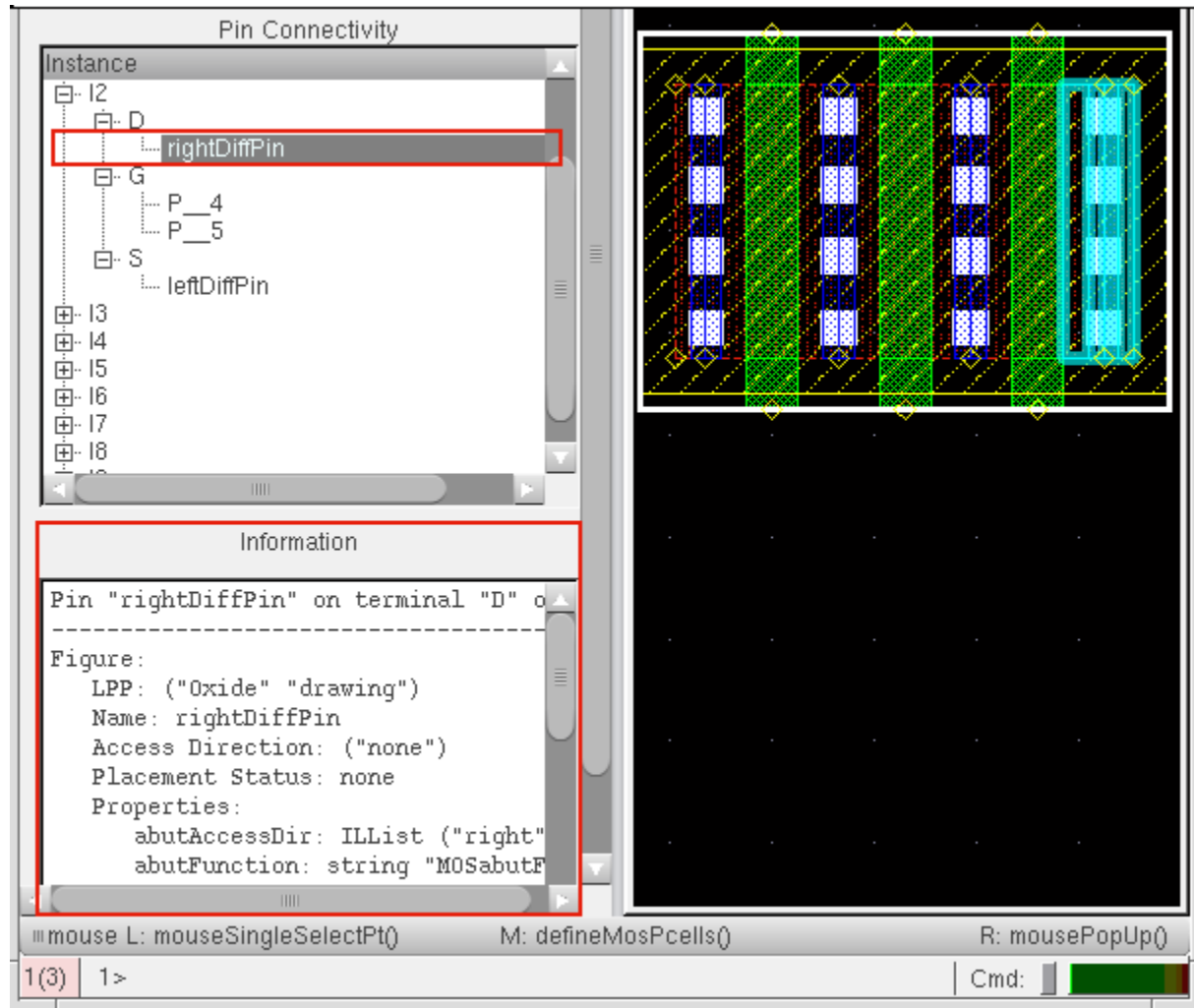


Strong Connect

If you select a terminal pin from the instance tree, the corresponding terminal pin in the cellview is highlighted in blue. Also, the pin information is displayed in the *Information* area, as shown in the figure below.

Virtuoso Parameterized Cell Reference

Debugging SKILL Pcells



- **Show 'Physical Only' Terminals:** You can use this option to view the list of logical terminals or all terminals including terminals with the "physical only" (`physOnly`) database attribute set for each instance. As shown in Figure 1, only logical terminals are listed under **I2** with this check box is not selected. However, the "physical only" attributes are added to the list if this check box is selected, as shown in the Figure 2.

Virtuoso Parameterized Cell Reference

Debugging SKILL Pcells

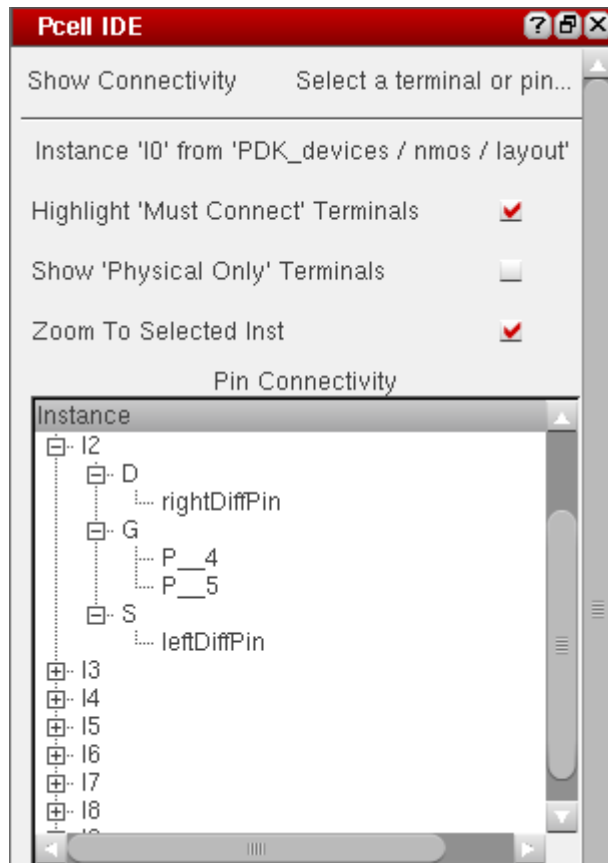


Figure 1

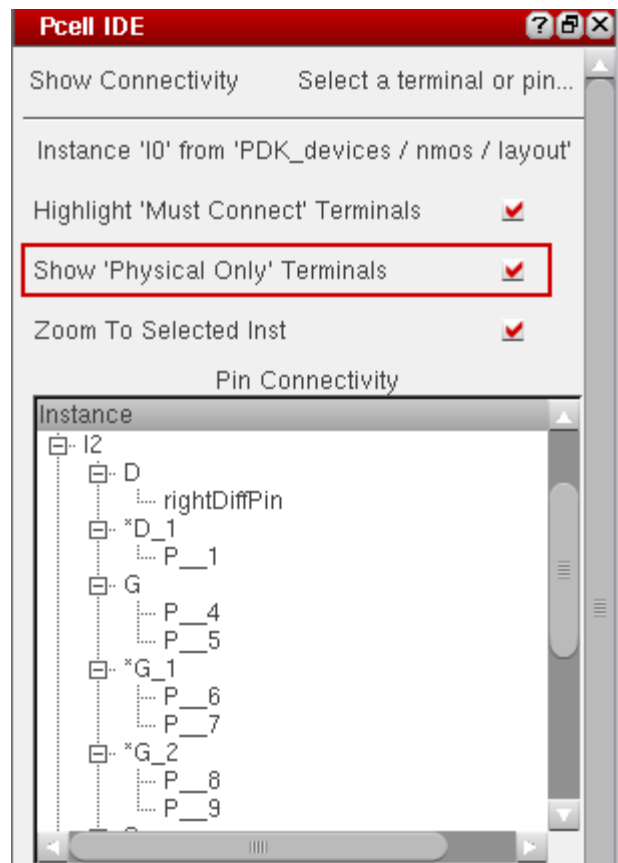


Figure 2

Note: The “physical only” (`physOnly`) attributes are indicated with an asterisk “*” in front of the terminal name.

- **Zoom To Selected Inst:** You can use this option to zoom in the selected instance or view the entire cellview.

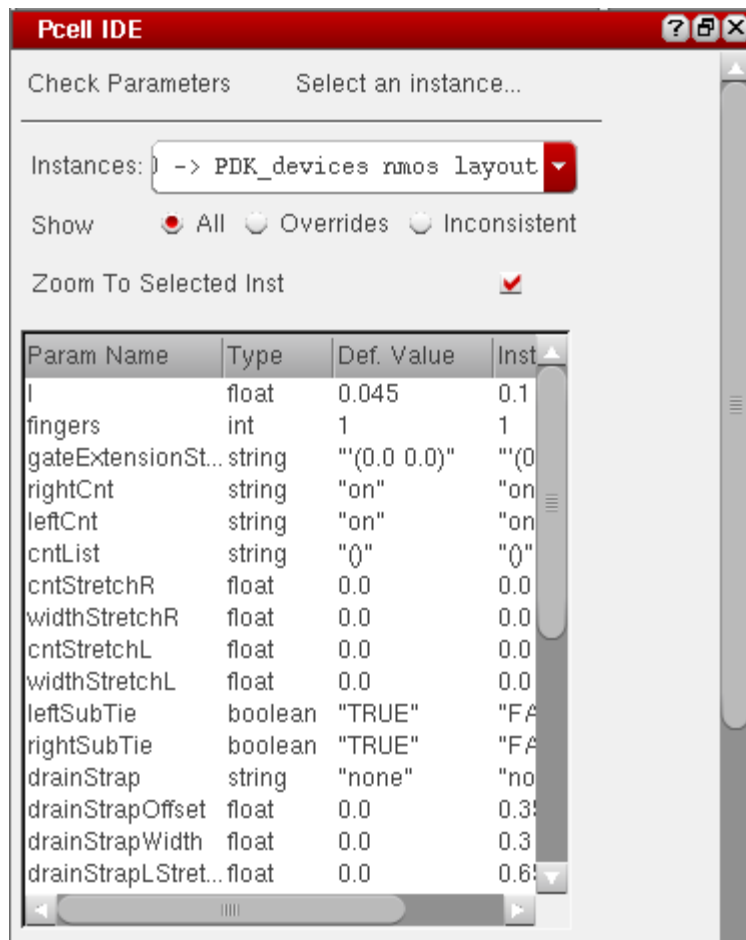
Checking Pcell Parameters

The *Check Parameter* command allows you to display the selected instance’s Pcell parameters, such as parameter name, type, default value, and instance value.

Note: This command is supported only for the top level design. Only top level instances are selectable in this command.

Virtuoso Parameterized Cell Reference

Debugging SKILL Pcells



In addition, you can use any of the following options to categorically display the list of the selected Pcell instance's parameter.

- **All:** You can use this option to display all parameters of the selected instance.
- **Overrides:** You can use this option to display the parameters whose value has been overridden on the instance.
- **Inconsistent:** You can use this option to display the parameters that have the inconsistency between its default value and CDF default value (if applicable).

Correcting and Updating Pcell Source

You can update the Pcell Source either from within the SKILL IDE source panel or from an external text editor. The updated source should then be loaded in SKILL IDE for it to be available for Pcell IDE. After you load the Pcell Source, the Debug Pcell form is displayed. To repeat the debugging process, you need to follow the procedure as described above.

Note: It is recommended to run SKILL Lint on the Pcell Source and correct all reported errors prior to loading it in SKILL IDE.

Restarting a Debugging Process

After the debugging process is completed, you can restart the debugging process by reloading a pre-loaded or a new Pcell Definition file. To restart a debugging process, the following options are available:

- You can restart the last evaluation with no change to the setup.
- You can modify parameter values for the same supermaster.
- You can select a new supermaster and set its parameter values.

Debugging Pcells Abutment

Abutment allows cells to be automatically overlapped, aligned, and electrically connected without introducing a design rule violation or connectivity error. Abutment reduces both the area occupied by a circuit and the length of the interconnect wiring. You can use abutment during interactive layout generation. Debug abutment is available as a plugin in VLS XL and is also integrated with Pcell IDE.

In VLS XL, when a command triggers device abutment, it leads to a debug abutment session. It allows you to debug all the events in VLS XL, VLS GXL (in IC6,1.8) and VLS EXL (in ICADV18.1). The Debug Abutment plugin is launched as an assistant that allows you to specify a set of events as breakpoints in a user-defined abutment function.

For more information on debug abutment in VLS XL, refer to the *[Debugging Abutment](#)* section in *Virtuoso Layout Suite XL User Guide*.

The debug abutment feature is integrated with Pcell IDE, which enables you to simulate the Pcell abutment without using the connectivity engine of VLS XL and execute the user-defined abutment functions. In addition, it allows you to debug Pcell evaluation that is triggered by the code that modifies the Pcell parameter inside the Pcell abutment function.

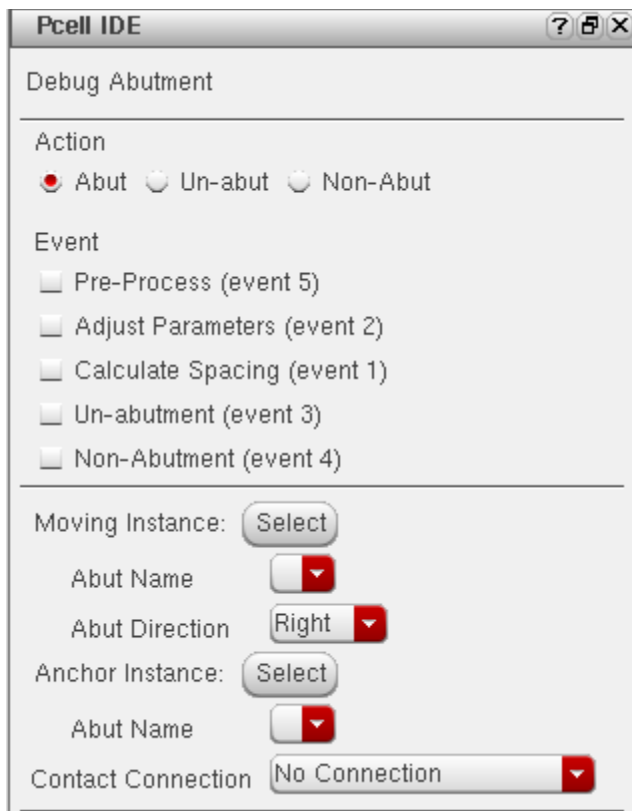
Debug Abutment Form

This functionality provides you with the ability to debug abutment using the Pcell IDE window. To do this, once you open the Pcell IDE window, you need to click the *Debug Abutment* icon from the Pcell IDE toolbar. The Debug Abutment form will be displayed.

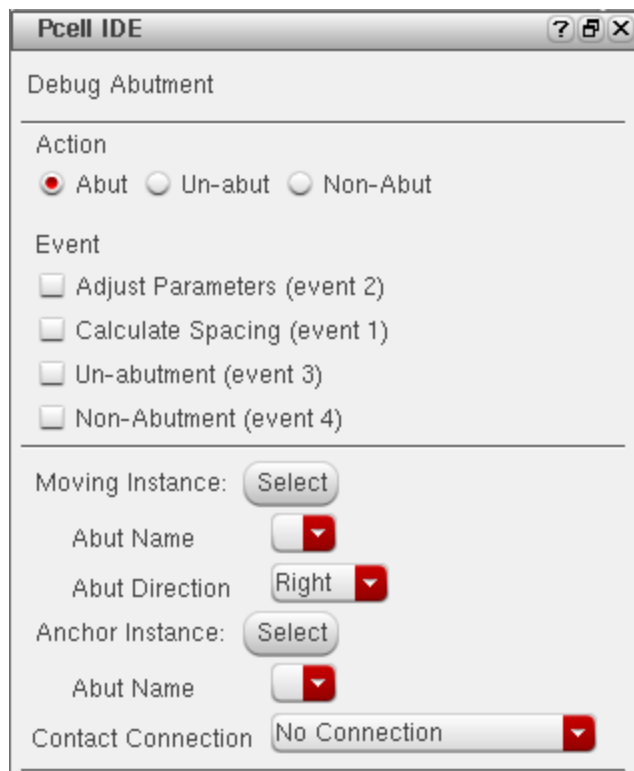


Virtuoso Parameterized Cell Reference

Debugging SKILL Pcells



The Pre-Process (event 5) option is available with the Advanced Node technology



The Pre-Process (event 5) is not available in the IC6.1.x release.



Caution

Only one instance of the Debug Abutment plugin can exist in one session. If you try to run multiple instances, the following message will be displayed.

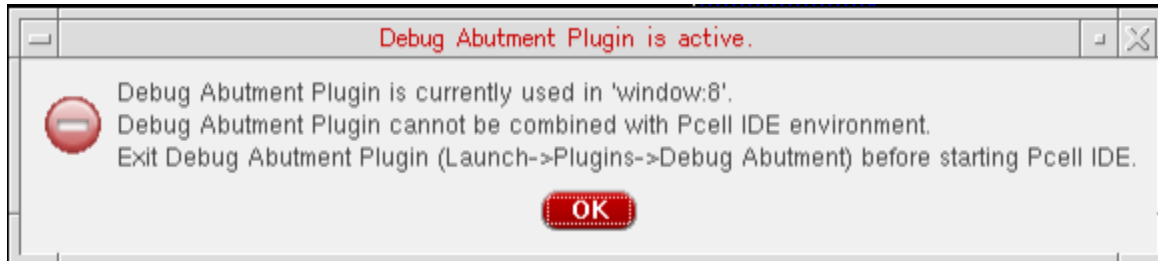


In addition, Debug Abutment Plugin and Pcell IDE cannot coexist in the

Virtuoso Parameterized Cell Reference

Debugging SKILL Pcells

same session. If you try to run Pcell IDE plugin when Debug Abutment plugin is already running, the following message will be displayed.



Using this functionality, Pcell IDE will trigger the abutment function when the basic requirements, such as abut class and abut figures' layers for abutment, are met.

Debugging Pcell Abutment using Pcell IDE

To debug Pcell abutment, you need to perform the following steps:

1. From the Action section, choose any of the following options:

- ☐ *Abut* – The abut simulator starts the abutment simulation. Debugging can be performed using the available abutment events.
- ☐ *Un-abut* – The abut simulator starts the un-abutment simulation. Once you select this option, all the check boxes under the Event section will be disabled and the *Un-abutment (event 3)* check box will be selected automatically. This is because the simulator will issue only the un-abutment event.
- ☐ *Non-Abut* – The abut simulator starts the non-abutment simulation. Once you select this option, all the check boxes under the Event section will be disabled and the *Non-Abutment (event 4)* check box will be selected automatically. This is because the simulator will issue only the non-abutment event.

Note: In case any of the *Un-abut* or *Non-abut* radio buttons is selected, then it does not show any selected event under Abut.

2. From the Event section, choose a set of events:

- ☐ *Pre-Process (Event 5)*: Pre-processes Pcells with Advance Node technology. This event is disabled when Pcells are not using the Advance Node technology. (ICADVM18.1 only)
- ☐ *Adjust Parameters (Event 2)*: Adjusts Pcell parameters for abutment.

Virtuoso Parameterized Cell Reference

Debugging SKILL Pcells

- ❑ *Calculate Spacing (Event 1)*: Calculates the offset between two abutted Pcells.
- ❑ *Un-abutment (Event 3)*: Adjusts Pcell parameters to unabut Pcells.
- ❑ *Non-Abutment (Event 4)*: Computes the spacing between cells that cannot abut.

Click the *Select* button corresponding to *Moving Instance* and then select the moving Pcell instance from the layout area. The selected moving instance will be highlighted in the layout area.

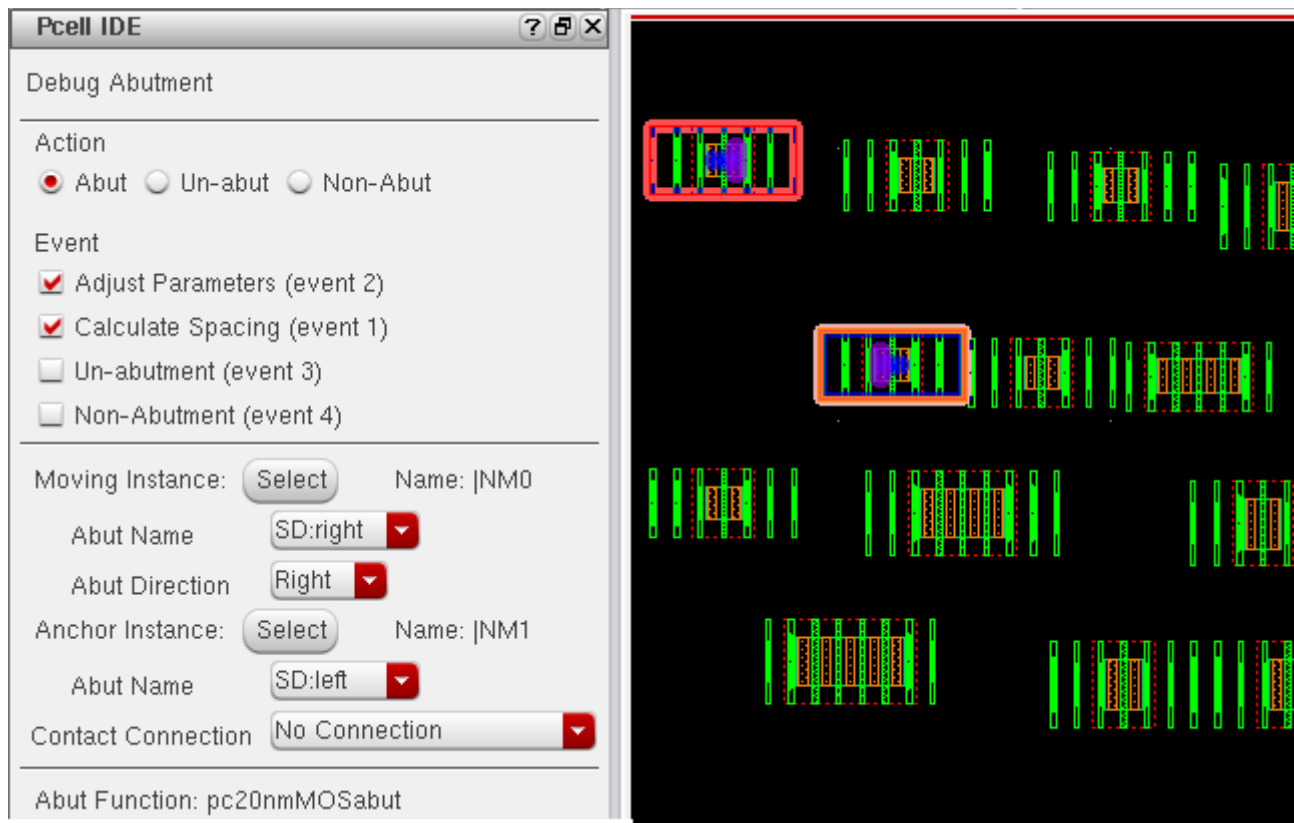
3. The abut figure names associated with the selected moving instance are listed in the *Abut Name* drop-down list. Select the appropriate abut name and direction from the *Abut Name* and *Abut Direction* drop-down list respectively.

Note: The abut name is either the abut figure's `abutFigName` or `pinFigName`.


4. Similarly, click the *Select* button corresponding to *Anchor Instance* and then select the anchor Pcell instance from the layout area. The selected moving and anchored instances are highlighted in the layout area.
5. The abut figure names associated with the selected moving instance are listed in the *Abut Name* drop-down list. Select the appropriate abut name from the *Abut Name* drop-down list, as shown in the figure below.

Virtuoso Parameterized Cell Reference

Debugging SKILL Pcells

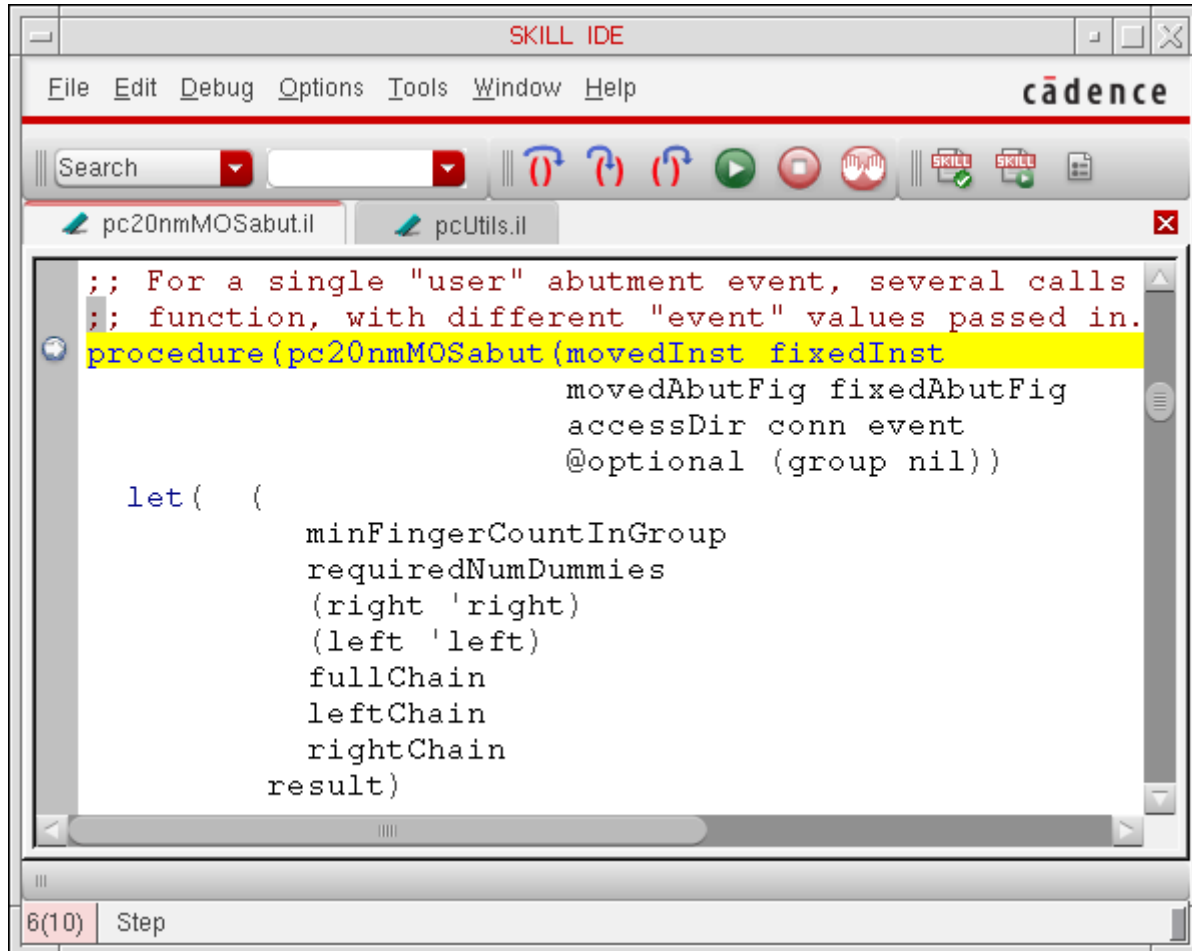


Note: The highlight colors of the moving and anchor abut figures are different.

6. Choose the abutment contact connection type from the *Contact Connection* drop-down list. It contains the following options:
 - ☐ *No Connection* – In case there is no connectivity.
 - ☐ *No External Connection* – The connectivity is only between the two abut figures involved.
 - ☐ *External Connection* – The connectivity is between the abut figures and to external connections.
7. Now, click the *Run*  icon from the Pcell IDE toolbar to start the debugging process.
8. The corresponding Abutment function code is displayed in the SKILL IDE window. You can walk through the code using the *Next*, *Step*, and *Continue* icons, as shown below.

Virtuoso Parameterized Cell Reference

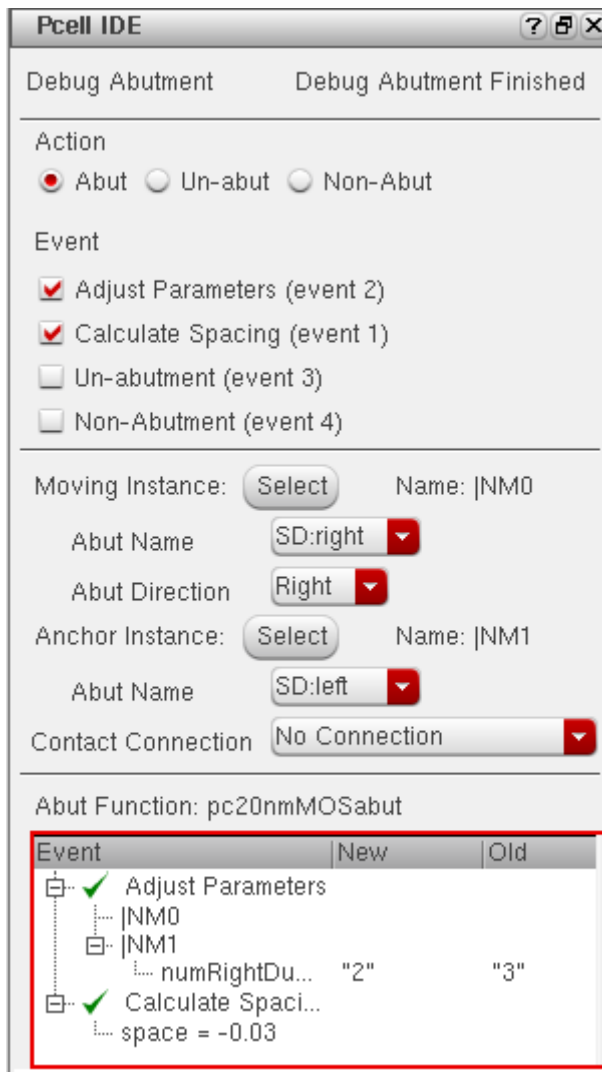
Debugging SKILL Pcells



9. After the debugging process is completed, the debug summary is displayed in the Pcell IDE window.

Virtuoso Parameterized Cell Reference

Debugging SKILL Pcells



As shown in the figure above, the result of each event is displayed in the Simulation Result Area of the Pcell IDE window. The green check mark icon indicates that the result of an event is true. Whereas, the red cross icon indicates that the result of an event is false.

- After the Adjust Parameters event is completed, the modified parameter is displayed with its old value, 3 and the new value, 2.
- After the Calculate Spacing event is completed, the calculated space value is displayed.

Note: A result for each event is displayed in a sequence as an event tree.

Virtuoso Parameterized Cell Reference

Debugging SKILL Pcells




Video

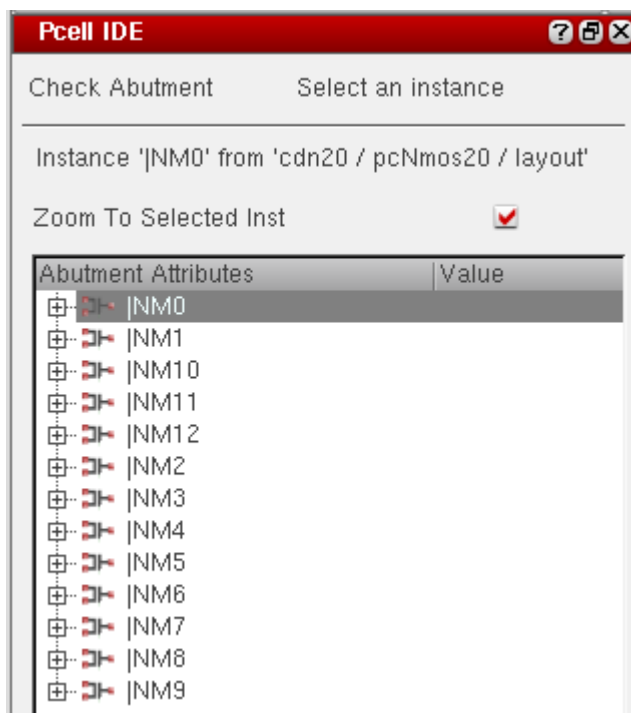
To view the demonstration on debugging abutment, see [Debug Abutment using Pcell IDE](#) video.

Note: Access to this video will depend on the availability of a web browser and a Cadence Online Support account.

Note: To debug the Pcell evaluation during debug abutment, you can turn on the Debug Hierarchy Pcell mode by clicking the *Debug Hier Mode* icon when any one of these functions — `dbReplaceProp`, `dbReplacePropList`, `dbDeletePropByName`, or `dbReplaceInstParamList` is called.

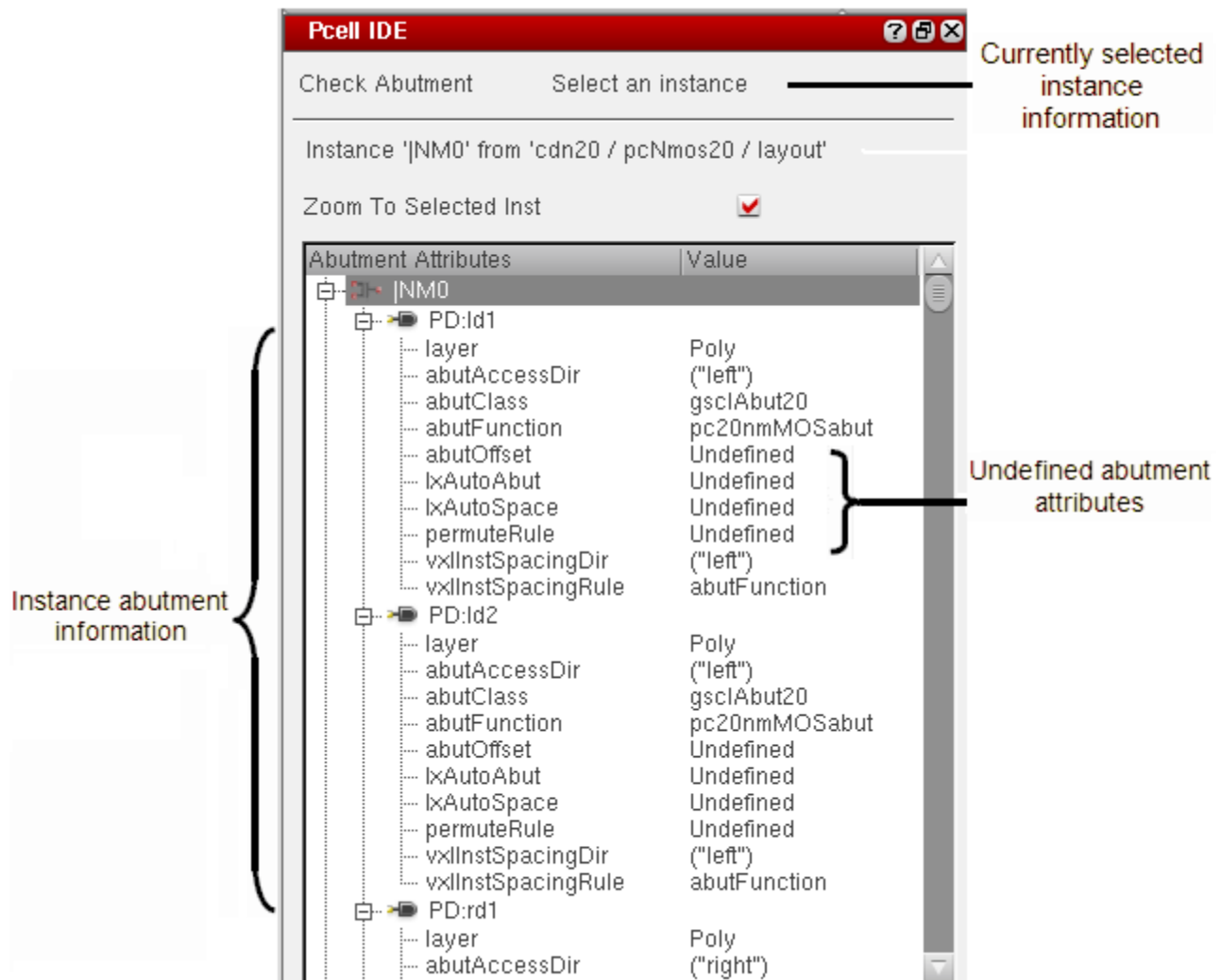
Check Abutment Command

The Check Abutment command enables you to analyze the abutment information that is stored in the database. To open the Check Abutment window, you need to select the *Check Abutment*  icon on the Pcell IDE toolbar.



You can select the specific instance to view its abutment information, such as the abut pin/figures information, as shown below.

Virtuoso Parameterized Cell Reference Debugging SKILL Pcells



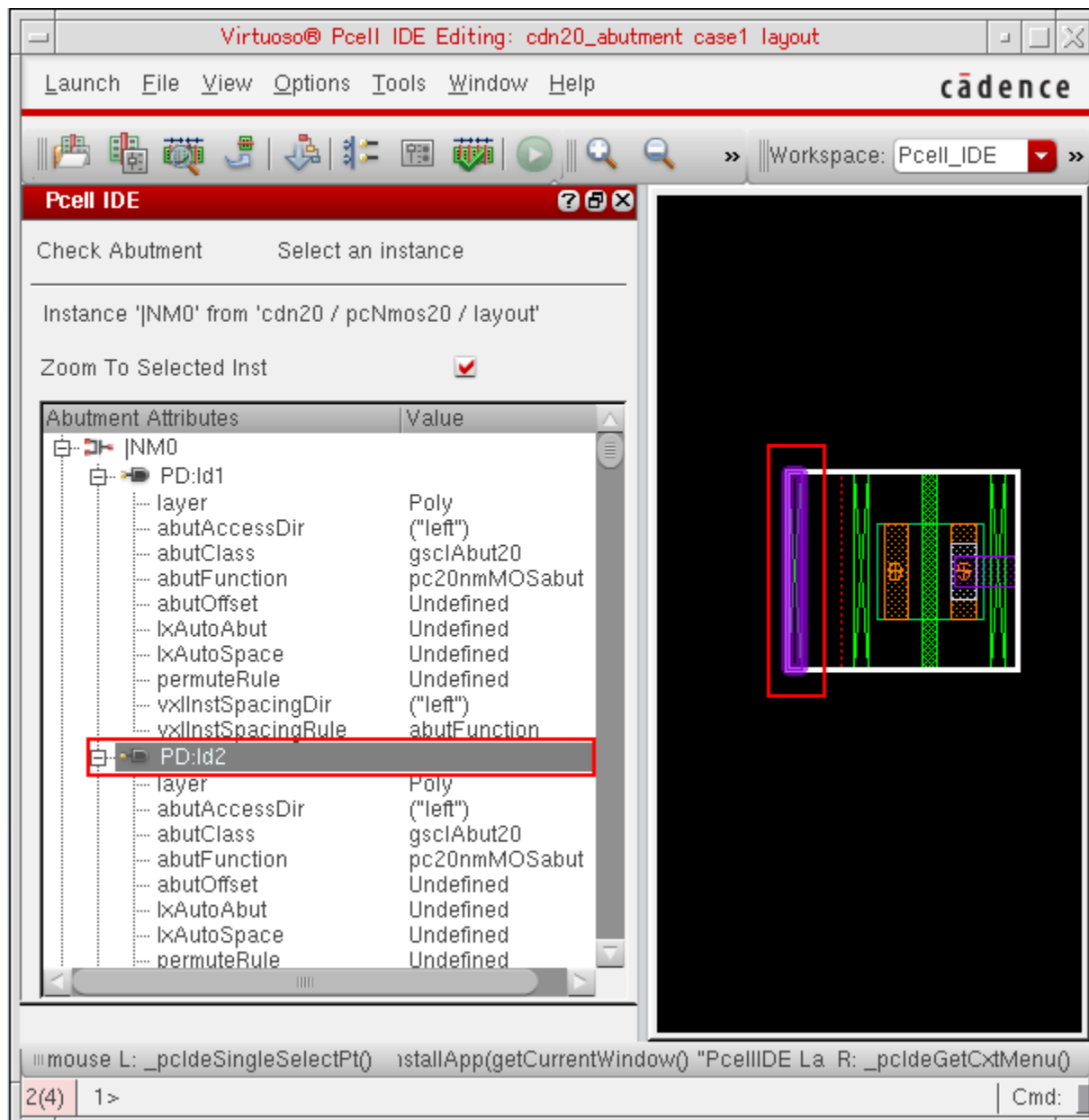
Notice that the following abutment attributes are listed in the Check Abutment window.

- *layer*
- *abutAccessDir*
- *abutClass*
- *abutFunction*
- *abutOffset*
- *lxAutoAbut*
- *lxAutoSpace*

Virtuoso Parameterized Cell Reference Debugging SKILL Pcells

- *permuteRule*
- *vxInstSpacingDir*
- *vxInstSpacingRule*

Once you select any of the instances, the corresponding instance will be highlighted and displayed in the layout area, as shown below.



Virtuoso Parameterized Cell Reference

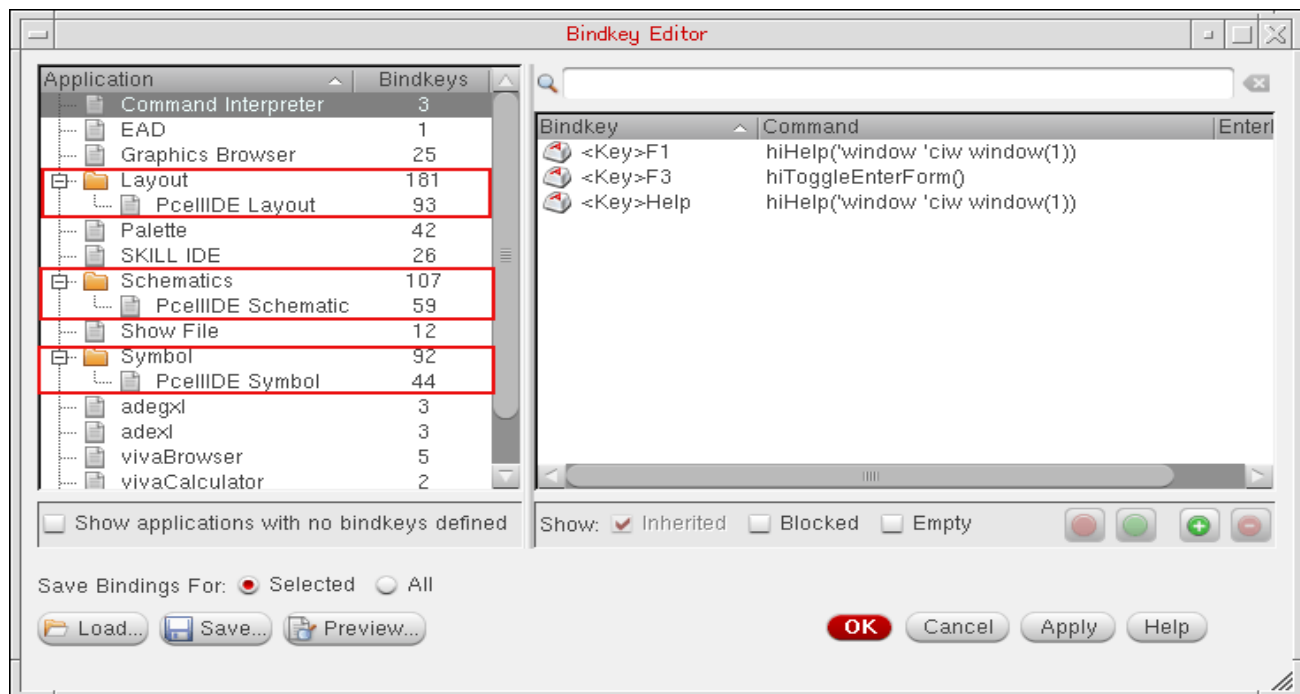
Debugging SKILL Pcells

Similarly, when you select any instance in the layout area, the corresponding instance name will be selected in the Pcell IDE window.

Note: If there is any missing abutment information on any abutment attribute, the attribute will appear as `Undefined` in the Value column.

Bindkey Editor

Pcell IDE has three sets of Bindkeys defined for the three Pcell IDE sub-applications, which includes PcellIDE Layout, PcellIDE Schematic, and PcellIDE Symbol. Therefore, in the Bindkey Editor, it shows these three sub-applications at the root level, as shown below.



Express Pcells

This chapter covers the following topics:

- Overview of Express Pcells
- Express Pcell Plug-In for Non-Virtuoso and Third Party Applications
- Requirements for Using the Express Pcell Feature
 - ❑ Virtuoso Design Environment Application
 - ❑ Non-Virtuoso Cadence Application
 - ❑ Third Party Application
- Use Model of Express Pcell Management
 - ❑ Inside the Virtuoso Design Environment
 - ❑ Outside the Virtuoso Design Environment
- Managing the Express Pcell Cache
 - ❑ Using Environment Variables
 - ❑ Using the Express Pcell Manager GUI in a Virtuoso Session
 - ❑ Maintaining the PDK Version in the Express Pcell Cache
 - ❑ Enabling Express Pcells for Specific Libraries
 - ❑ Installing the Express Pcell Plug-In for Non-Virtuoso or Third Party Applications
- Cache Merge Utility
- Environment Variable to Control Express Pcell Proliferation Message
- Exclude Specified Pcells from the Express Pcell Save or Read Operation

Overview of Express Pcells

Traditional SKILL Pcell submasters, once evaluated, reside in memory and are not saved to disk (see [Creating SKILL Parameterized Cells](#) for more information). This means that the Pcells must be evaluated every time a design is opened, which can become a performance bottleneck in designs that contain a large number of complex Pcell instances.

The Express Pcell feature, which is available in the Virtuoso environment, maintains a cache of pre-evaluated Pcells on disk, meaning that Pcell code does not need to be evaluated every time a design is opened. This offers a significant performance boost not only for designs that contain a large number of Pcell instances, but also for designs with modest numbers of instances of complex Pcells involving computationally intensive SKILL code.

Note: Express Pcell does supports both layout and schematic (and symbol) Pcells. For more information, contact Contact Cadence Customer Support.

Express Pcell Plug-In for Non-Virtuoso and Third Party Applications

Pcells in the Virtuoso design environment are predominantly written using SKILL and are evaluated using a SKILL Pcell evaluator that is not available outside the Virtuoso design environment. This means that these SKILL Pcells cannot easily be read or used outside the Virtuoso design environment. Non-Virtuoso Cadence applications and third-party applications rely on other alternatives for SKILL Pcell evaluation. This alternative data flow channel between the applications and OpenAccess can be time-consuming and not entirely seamless.

The Express Pcell feature is available on OpenAccess as a SKILL Pcell plug-in (*.so and *.plg files). The plug-in leverages the interoperability offered by OpenAccess to allow non-Virtuoso Cadence and third party applications to access Express Pcell generated data saved on OpenAccess without having to translate it. The plug-in enables these applications to access the pre-evaluated Pcell submasters resulting in seamless read-interoperability of SKILL Pcells. The plug-in also provides some limited write-interoperability, so that you can edit the container of Pcell instances. For example, third party applications (such as a router or a verification tool) can open a Virtuoso-generated OpenAccess design safely and add markers or perform routing in them directly.

Note: All Express Pcell data is backward compatible. So, for example, IC6.1.8 can read any pre-existing cached data generated using an earlier version of the Express Pcell engine. However, the cached Express Pcell data generated using IC6.1.8 cannot be read by a previous version of the software or plug-in.

Note: Express Pcell plug-in does not support schematic (and symbol) Pcells.

Requirements for Using the Express Pcell Feature

Virtuoso Design Environment Application

To use the Express Pcell feature in the Virtuoso design environment, you require:

- At least OpenAccess version OA 22.04.001
- Cadence Design Framework II (License Number 111)
- The [CDS_ENABLE_EXP_PCELL](#) variable to be set to `true`.

If the required license is not available or cannot be checked out, the Express Pcell feature is disabled for that particular Virtuoso session.

Note: If the SKILL code associated with a Pcell supermaster evaluation function creates or initializes some global data that is to be used elsewhere within the same Virtuoso session, the use of Express Pcells in the same session might result in some unexpected behavior. This is because the associated SKILL code is not executed in the presence of the Express Pcell cache. An example of such a scenario is the creation or initialization of a global variable in some Pcell evaluation function and the subsequent use of that global variable in its corresponding Pcell abutment function. Consequently, when using the Express Pcell feature in a Virtuoso session, take care to organize the evaluation code of your Pcells to avoid the use of global variables. Alternatively, avoid the creation or initialization of global variables outside the Pcell evaluation code when using the Express Pcell feature in a Virtuoso session.

Non-Virtuoso Cadence Application

A non-Virtuoso Cadence application checks out the VLS L license.

Third Party Application

Third party applications check out the VLS L license. The following files are required for the successful use of the plug-in by any third party application:

- `cdsSkillPcell.plg` file
- `libcdsSkillPcell.so` file

Note: Any third party applications intending to load or use the Express Pcell feature must be compatible with the generated `.so` libraries in the Virtuoso release in question.

Virtuoso Parameterized Cell Reference

Express Pcells

For information about how to install the plug-in, see [Installing the Express Pcell Plug-In for Non-Virtuoso or Third Party Applications](#).

Important

The Express Pcell plug-in library, `libcdsSkillPcell.so` is compiled against `libstlport.so`. This can lead to drop-in compatibility issues if used with an OpenAccess application compiled against `libcstd.so`. To avoid such issues while using the plug-in's shared library with a non-DFII OpenAccess application, both the plug-in's shared library and the application must be compiled against the same C++ library, `libstlport.so` or `libcstd.so`. The following additional 64bit plug-in library version is also available for non-DFII OpenAccess applications compiled with the standard C++ library (`libcstd.so`):

`cdsSkillPcell/lib/64bit/libcdsSkillPcell_nonSTLport.so`

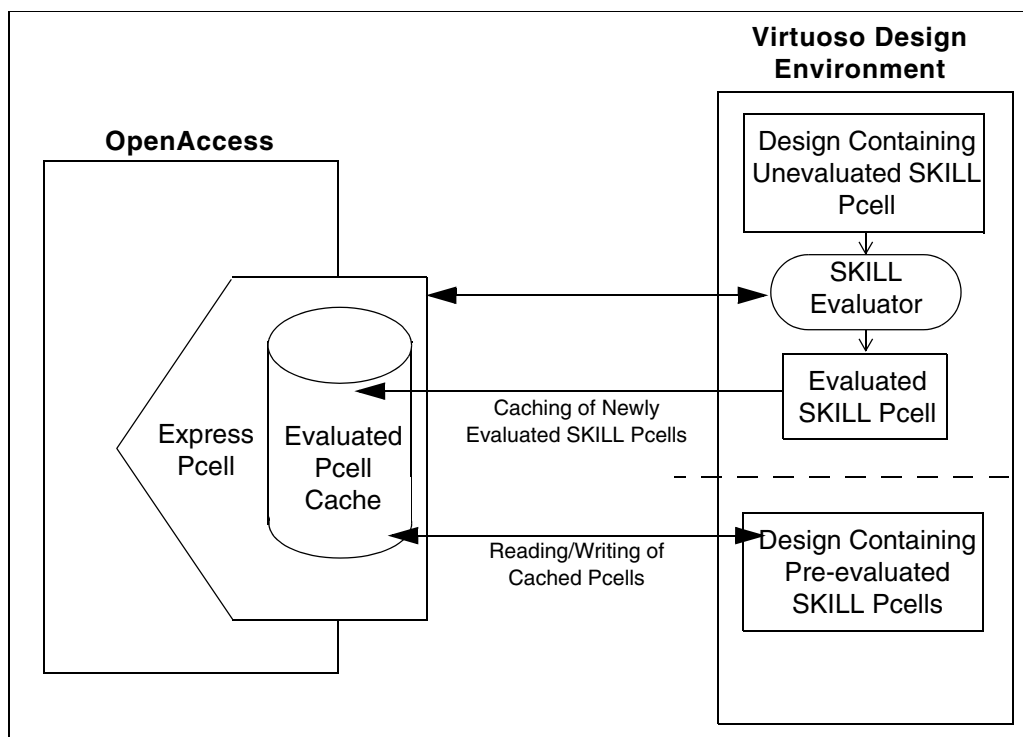
These plug-in libraries are available in the Express Pcell Plugin Kit. To use these libraries, you must set up the environment by changing the soft link in either `tools/lib` or `tools/lib/64bit` to the corresponding non-stlport library available in `tools/cdsSkillPcell/lib`. If you do not have permission to change the link in `<install_dir>/tools/lib`, you can create a soft link for `libcdsSkillPcell.so` ==> `<install_dir>/tools/cdsSkillPcell/lib/libcdsSkillPcell_nonSTLport.so` in a separate directory and specify that directory in the `LD_LIBRARY_PATH` to be read by the non-DFII OpenAccess application.

Use Model of Express Pcell Management

This section explains how applications both inside and outside the Virtuoso design environment use the Express Pcell cache.

Inside the Virtuoso Design Environment

The following figure shows how Virtuoso design environment applications use the Express Pcell cache.



When you open a cellview containing a Pcell instance for the first time, a Pcell submaster is created in memory by the SKILL Pcell evaluator.

When requested by the OpenAccess Pcell evaluator to populate the submaster of a Pcell instance, the Virtuoso SKILL Pcell evaluator sends a request to the Express Pcell framework, which in turn checks for the existence of the requested submaster in the cache. If the submaster exists, it is read from the cache. If it does not exist, it is created by the usual SKILL Pcell evaluation process in the Virtuoso design environment.

You can save the evaluated submasters using the *File – Save* command (see [Auto Save](#)). This generates a cache of the Pcell submasters in the specified Express Pcell directory ([CDS_EXP_PCELL_DIR](#)).

Virtuoso Parameterized Cell Reference

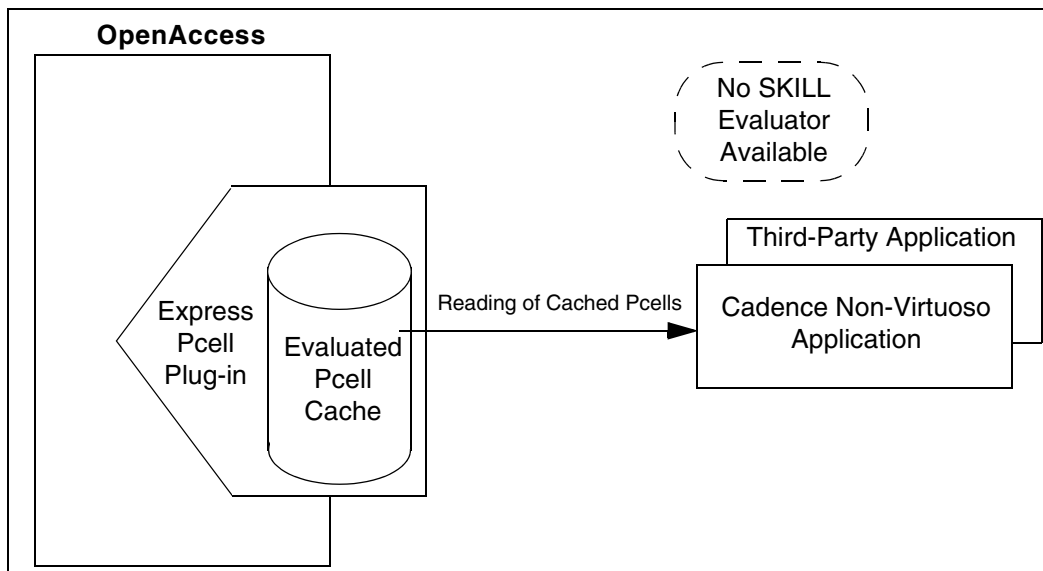
Express Pcells

You can also cache the Pcell submasters that are available in memory by clicking Save Cache in the Express Pcell Manager form.

Note: If a Pcell supermaster is subsequently updated on disk, the Save/Update operation on the cache (through *File – Save* in the design or by using the Express Pcell Manager form) refreshes the cache of the corresponding submasters on disk.

Outside the Virtuoso Design Environment

The Express Pcell plug-in provides read access to the cache data to non-Virtuoso Cadence applications as well as third party applications on OpenAccess. In this way, these applications can also access the submasters already evaluated by Virtuoso design environment applications and stored in the cache.



This model provides seamless read interoperability of SKILL Pcells to non-Virtuoso Cadence and third party applications.

Managing the Express Pcell Cache

By default, intermediate Express Pcell files are stored in a `.expressPcells` directory, which is created inside the current working directory. This directory maintains a cache of pre-evaluated Pcells on the disk. This means that the Pcell code does not need to be evaluated every time a design is opened. This offers a significant performance boost not only for designs that contain a large number of Pcell instances but also for designs with modest numbers of instances of complex Pcells involving computationally intensive SKILL code.



Advanced node and mature node caches are compatible. However, IC6.1.6 and Advanced Node will not provide any backward portability for static Virtuoso. Therefore, cache generated using IC6.1.5 will not open in IC6.1.6 or Advanced Node. However, IC6.1.6 and Advanced Node plug-in will be able to read the IC6.1.5 generated cache, and provide backward compatibility. When a IC6.1.5 generated cache is encountered in IC6.1.6 or Advanced Node, Virtuoso Express Pcell will be disabled and the error message will be displayed.

Using Environment Variables

Use the following shell environment variables to manage the Express Pcell cache. These variables are read during Virtuoso startup and must therefore be set before you launch the application.

■ CDS_ENABLE_EXP_PCELL

Controls the caching of evaluated SKILL Pcells. The default value of this variable in a Virtuoso session is different from its default value when used with a non-Virtuoso or third party application.

In a Virtuoso session

By default, this variable is set to `false` in the Virtuoso environment, which means that SKILL Pcells are evaluated every time a design is loaded in a Virtuoso session. To use the Express Pcell feature, set this variable to `true` before starting the session.

In a non-Virtuoso session

The default value of this variable is `true` when the Express Pcell plug-in is used with a non-Virtuoso or third party application. Assuming that the plug-in files (`.plg` and `.so`) are correctly installed (see [Installing the Express Pcell Plug-In for Non-Virtuoso or Third Party Applications](#)), available Pcell submasters are automatically read from the cache

Virtuoso Parameterized Cell Reference

Express Pcells

during an OpenAccess Pcell evaluator call. You can change this default behavior by setting the variable to `false` before starting the non-Virtuoso or third party application.

■ CDS_EXP_PCELL_DIR

Specifies the directory to be used for caching Pcells. You can specify either an absolute path or a path relative to your current working directory.

Using this environment variable allows multiple users or Virtuoso sessions to point to the same Express Pcell directory for concurrent read/write access. This helps in maintaining a single central Express Pcell cache for a team of layout designers working on a single design project, thereby avoiding duplicate or inconsistent cached Express Pcell data in any private directories. In addition, it ensures the availability of the complete cache at a single location and therefore avoids the need to regenerate the entire Express Pcell cache for the entire chip or block.

Note: The recommended umask value is 022, which masks out write permission for other users and ensures that only the directory owner has permission to add new files. However, if the Express Pcell cache directory is to be used by multiple users one at a time or is being accessed concurrently in a multi-user shared cache scenario, then it is recommended to use a unmask setting of 002, which means the entire group has permission to write to the directory.

■ CDS_EXP_PCELL_MISMATCHED_PDK_POPUP

Controls the display of a pop-up window when `xPcell_Version_PDK` does not match the version found in the existing cache. When this environment variable is set to `true`, a pop-up window opens informing you about the version mismatch.

■ CDS_EXP_PCELL_RDONLY

Controls the ability to save, clear, and update cached Pcell data.

If set to `true`, the cached Pcell data can be read but not updated.

If set to `false` (the default), and you have write permission for the cache directory, you can update the cache data as desired. If you do not have write permission, you will see a warning message and the cache data remains available for reading.

Note: You are not expected to manually update or delete the files in the Express Pcell directory. If a Pcell data file is explicitly deleted from the Express Pcell directory (for example, by using the `rm` command in a terminal window), then the lost contents of this file will not be generated automatically while saving the cache. However, in this case, the lost contents will be regenerated using the Update Cache feature. Therefore, it is always recommended to delete or update the files in the Express Pcell directory using the Express Pcell Manager (Clear Cache/Save/Update).

Virtuoso Parameterized Cell Reference

Express Pcells

■ CDS_EXP_PCELL_NO_MULTUSER

Disables the multi-user support feature of the Express Pcell cache. It is recommended that you set this variable on systems using the Andrew File System (AFS), because the Express Pcell infrastructure does not provide multi-user support on AFS.

■ CDS_XPCELL_LIB_EXCLUSION

Excludes from Pcell cache operations those PDK libraries that do not have the `xPcell_Enable_libName` variable defined. By default, such libraries are automatically included to maintain backward compatibility. See [Enabling Express Pcells for Specific Libraries](#) for more information.

■ CDS_XPCELL_LIB_FILTER_FILE

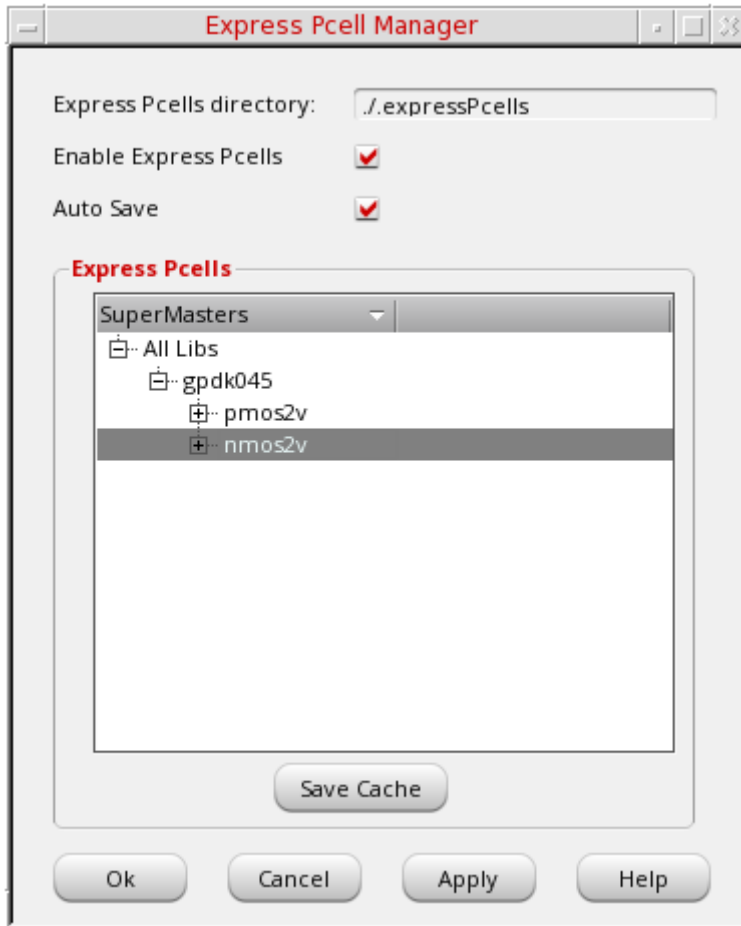
Specifies a user-defined library filter file that lets you flexibly override the `xPcell_Enable_libName` variable settings in the `libInit.il` file for a given library. This gives you full control over which libraries are included in Pcell cache operations and which are excluded. See [User Control to Override the Inclusion and Exclusion of Libraries](#) for more information.

Using the Express Pcell Manager GUI in a Virtuoso Session

You can manage the Pcell cache using the Express Pcell Manager graphical user interface.

1. From the layout window menu bar, select *Tools – Express Pcell Manager*.

The Express Pcell Manager window is displayed.



Express Pcells directory is a read-only field showing the Pcell cache directory.

You can specify a different location for storing the Pcell cache using the [CDS_EXP_PCELL_DIR](#) variable.

However, a new directory will be created if the current working directory is moved or deleted by the user in mid session.

If the cache directory is not writable, the cache is not saved and an appropriate message is displayed.

Note: When the cache is disabled, all indexes are properly cleared.

Virtuoso Parameterized Cell Reference

Express Pcells

2. Check the Enable Express Pcells box to enable the Express Pcell feature.

Deselect the option to revert to conventional SKILL Pcell evaluation.

The default setting controlled by the [CDS_ENABLE_EXP_PCELL](#) variable.

Note: The values of the `CDS_EXP_PCELL_DIR` and `CDS_ENABLE_EXP_PCELL` variables are read during the Virtuoso session startup.

3. Check the Auto Save box to automatically save any updated Pcell variants when the design is saved.

If you deselect this option, you will be prompted to save any updated Pcell variants each time you save the design.

4. Use the right mouse button in the *Express Pcells* pane to perform the following operations on selected libraries or cellviews:

- ☐ Ignore Timestamp
- ☐ Update Cache
- ☐ Clear Cache

Notes:

1. Cached Pcell data is updated even when referenced data, such as an attached technology library, is updated. However, such an update may not necessarily update the supermaster's timestamp. In such situations, you are advised to refresh the cache data on the disk by using the Express Pcell Manager.
2. You might need to save the design even if no changes are made to the design to allow the cached data to be updated on disk in the `.expressPcells` directory.
3. Opening just the top-level layout (Display Options *Stop Level* set to 0) for the first time does not evaluate Pcells in the hierarchy. Therefore, no cache data is created for them.

Note: The `.xpcCache` directory is automatically created with write permission inside the `.expressPcells` directory. Any user trying to read the cache should also have write permission in the `.xpcCache` directory.

Maintaining the PDK Version in the Express Pcell Cache

The Express Pcell cache is PDK version-aware and can notify users if the Pcells currently in the cache are out of date or not. This lets you keep the cache current whenever the PDK version is updated during a particular design cycle.

Virtuoso Parameterized Cell Reference

Express Pcells

To use this feature, the CAD engineer or PDK developer must define the following library version SKILL variable in the `libInit.il` of the PDK in question:

```
xPcell_Version_PDK = "version"
```

Where *PDK* is the name of the PDK in question and *version* is any unique string value used to identify the version. For example:

```
xPcell_Version_gpdK045 = "3.0"  
xPcell_Version_gpdK028 = "2.0a"
```

Important

The CAD engineer or PDK developer must ensure that the library version SKILL variable is updated for each new PDK release, otherwise Virtuoso will be unable to correctly recognize different versions.

The table below shows the behavior of each cache operation when `xPcell_Version_PDK` does not match the version found in the existing cache.

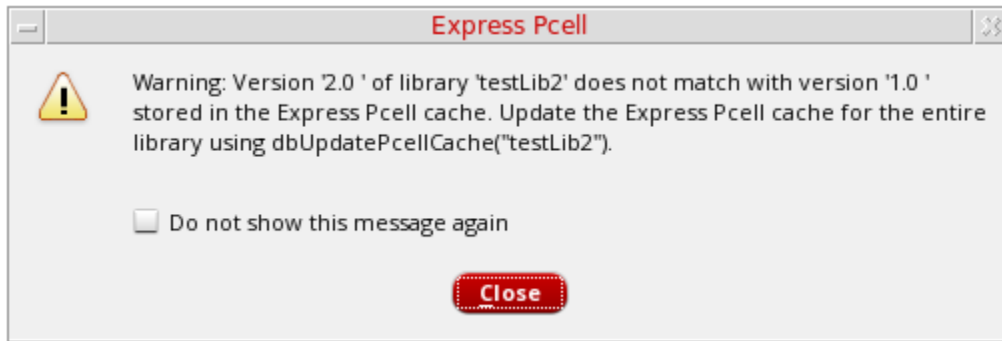
Operation	Behavior
Read cache	Cache not read (Pcell will be evaluated) CIW warning to update cache
dbSavePcellCache()	Cache not saved CIW warning to update cache
dbSavePcellCacheForCV()	Cache not saved CIW warning to update cache
dbSavePcellCacheForCVOnly()	Cache not saved CIW warning to update cache
dbUpdatePcellCache()	Cache updated
dbClearPcellCache()	Cache cleared

The `CDS_EXP_PCELL_MISMATCHED_PDK_POPUP` environment variable controls the display of a pop-up window when `xPcell_Version_PDK` does not match the version found in the existing cache. When this environment variable is set to `true`, the following pop-up window

Virtuoso Parameterized Cell Reference

Express Pcells

opens informing you the mismatch between the version of the library and the version of Express Pcell stored in the cache.



Select *Do not show this message again* to disable the pop-up window for the entire Virtuoso session, and close the window to proceed.

Note:

- When there is a version mismatch, one warning is generated for each PDK, not for each individual submaster.
- If the `xPcell_Version_PDK` variable is not defined for a PDK, the default is taken as " " (empty string).

Enabling Express Pcells for Specific Libraries

The Express Pcell framework lets you specify which libraries are included/excluded for cache operations.

This allows you to disable cache operations for dynamic reference libraries containing Pcells that are edited and optimized regularly but for which supermasters are not recompiled. Excluding such libraries from cache operations means that Pcells from these libraries are evaluated every time a design is opened, ensuring that the latest version of the Pcell is always used in the design in question.

To use this feature, the CAD engineer or PDK developer must define the following library inclusion SKILL variable in the `libInit.il` of the PDK in question:

```
xPcell_Enable_libName = t | nil
```

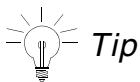
Where:

<code>libName</code>	The library name.
<code>t</code>	The library is included for cache operations.

Virtuoso Parameterized Cell Reference

Express Pcells

<code>nil</code>	The library is excluded from cache operations.
<code><undefined></code>	The library is included for cache operations.
	This ensures that the default behavior is not disrupted for libraries that do not have the SKILL variable set. Such libraries continue to be included in Express Pcell cache operations by default.



Tip

To reverse this behavior and exclude libraries that do not have the SKILL variable set from Pcell cache operations by default, set the following shell environment variable before you launch Virtuoso:

```
setenv CDS_XPCCELL_LIB_EXCLUSION
```

When set, libraries for which the SKILL variable is undefined are *excluded* by default.

The table below shows the behavior of cache operations when `xPcell_Enable_libName` = `nil` for a library.

Operation	Behavior
Read cache	Cache not read (Pcell will be evaluated)
<code>dbSavePcellCache()</code>	Cache not saved
<code>dbSavePcellCacheForCV()</code>	Cache not saved
<code>dbSavePcellCacheForCVOnly()</code>	Cache not saved
<code>dbUpdatePcellCache()</code>	Cache updated
<code>dbClearPcellCache()</code>	Cache cleared

User Control to Override the Inclusion and Exclusion of Libraries

You can override the value of the `xPcell_Enable_libName` variable for individual libraries by specifying a user-defined library filter file before you launch Virtuoso:

```
setenv CDS_XPCCELL_LIB_FILTER_FILE "filename"
```

Virtuoso Parameterized Cell Reference

Express Pcells

Where *filename* refers to a simple ASCII file of the following format:

```
;libName      value
gpdK045       t
ref_RC        nil
ref_L         nil
```

Only the libraries defined in the list are impacted. Where there is a conflict, the value specified in the user-defined filter file takes precedence.

Important

You must set the shell environment variable before invoking Virtuoso to obtain your required behavior.

Installing the Express Pcell Plug-In for Non-Virtuoso or Third Party Applications

The `cdsSkillPcell.plg` and `libcdsSkillPcell.so` files required for the installation of the plug-in on OpenAccess, are at the following locations:

- `cdsSkillPcell.plg` file: Description file of the plug-in as required by OpenAccess

Location: `<install_dir>/tools.<platform>/cdsSkillPcell/plugin/cdsSkillPcell.plg`

For Virtuoso and Innovus, the OpenAccess installation contains the `cdsSkillPcell.plg` file in the `<install_dir>/share/oa/data/plugins` directory.

- `libcdsSkillPcell.so` file: Shared library

Location: `<install_dir>/tools.<platform>/cdsSkillPcell/lib/libcdsSkillPcell.so`

For the successful use of the plug-in by any third party application, it is important to ensure the following:

- Availability of `cdsSkillPcell.plg` in the `<OA_Installation>/data/plugins` directory
- Inclusion of `libcdsSkillPcell.so` in the library search path, such as `$LD_LIBRARY_PATH`, of the respective operating system
- Inclusion of the path of `cdslmd` in `$PATH` (`cdslmd` exists in the `<install_dir>/tools/bin` directory)
- Express Pcell Plugin (`libcdsSkillPcell.so`) has a dependency on `libcdsCommon_sh.so` and `libcls_sh.so` shared libraries. These libraries are available in the `cdsCommon` kit or can be copied from the `<cadence_installation_hierarchy>/tools/lib` directory. These libraries must be present in the library search path (`LD_LIBRARY_PATH/LIBPATH`), which is visible to the application using the plug-in. You can also add `<cadence_installation_hierarchy>/tools/lib` in their library search path to achieve this.

Note: Some third party applications do not honor the external library search path (`LD_LIBRARY_PATH/LIBPATH`) because they define their own library search path internally. In such cases, you can copy the desired `libcdsCommon_sh.so` and `libcls_sh.so` shared libraries in the `<OA_HOME_Installation>/lib/<platform_port>/opt` directory. This ensures the smooth working of third party applications with the Express Pcell functionality.

Cache Merge Utility

The cache merge utility collects and merges the contents of multiple Pcell caches located in different directories into a single Pcell cache. This is useful when members of a project team working on the same set of PDKs and reference libraries work from multiple project sites. In such a scenario, this utility can be used to consolidate submasters stored at various locations into a single destination cache directory. This eliminates repetition of submaster data, thereby reducing the disk space used and the Pcell read time for individual team members.

The `cds.lib` file must contain entries for all of the reference libraries used by all the caches that are to be merged. However, where two or more caches are using the same reference library, you must ensure that the reference libraries are synchronized.

Preventing Data Inconsistencies

Data inconsistencies can arise in the following cases:

- The timestamps stored for a supermaster are different in different locations.

In this case, the cache merge utility does not merge the submasters of that supermaster and issues an error message stating that the data across multiple locations or directories is inconsistent and needs to be regenerated at each location by using the correct supermaster or PDK version.

Note: You can use the Express Pcell Manager to regenerate caches.

- The timestamps stored for a supermaster are the same, but different PDK versions are used across locations.

In this case, the cache merge utility cannot determine if different PDK versions are used. Therefore, you must ensure that you use the cache merge utility only for those projects that use the same PDK version across locations. You can also develop your own utilities or processes based on your data setup to track any such inconsistency.



Do not share the xPcell cache between different PDK versions. If you want to share the xPcell cache between different PDK versions, recompile/regenerate all the affected supermasters.

xpcmerge Command Line Utility

Use the `xpcmerge` command to merge the contents of multiple caches to a specified destination cache. If the destination Pcell cache contains any data, by default the merged data is appended to the destination cache data. You can choose to delete any existing data stored in the destination Pcell cache before writing the merged data to the new cache.

Note: This utility requires a `cds.lib` specifying the paths of PDKs and libraries. If the utility detects a PDK version mismatch for a Pcell, the corresponding supermasters and submasters are removed from the merged cache. If a PDK mismatch is detected for a library, no merge happens for that library and a warning message is issued.

Arguments

<code>-dest xPcellDir</code>	Specifies the name of the destination Pcell cache.
<code>-src xPcellDir1 [xPcellDir2 xPcellDirN ...]</code>	Specifies the names of multiple source Pcell caches.
<code>-log logFileName</code>	Specifies a relative or absolute path to a log file. See Sample Log File Format for details.
<code>-noDestMerge</code>	Deletes any existing data in the destination Pcell cache before the merged data is written to the destination Pcell cache.

Example 1

```
xpcmerge -dest /hm/user1/.expressPcell -src /hm/user2/.expressPcell
```

Appends the contents of the source Express Pcell cache `/hm/user2/.expressPcell` to the destination Express Pcell cache `/hm/user1/.expressPcell`.

Note: In this example, if you use the `-noDestMerge` option, then the `xpcmerge` command will exit and an error message will be displayed.

Virtuoso Parameterized Cell Reference

Express Pcells

Example 2

```
xpcmerge -dest /hm/user1/.expressPcell -src /hm/user2/.expressPcell ../india/.expressPcell -noDestMerge -log /hm/admin1/logs/cacheMerge.log
```

Merges the contents of the two source Pcell caches, `/hm/user2/.expressPcell` and `../india/.expressPcell`. Because the `-noDestMerge` option is specified, the contents of the destination directory, `/hm/user1/.expressPcell`, are deleted before the merged content is written to the destination directory.

In addition, the `/hm/admin1/logs/cacheMerge.log` file is created and the following information is written to it.

- Arguments passed to the cache merge utility
- Information about the supermasters and submasters
- Appropriate error and warning messages if the merge operation did not run successfully
- Merge completion status

Virtuoso Parameterized Cell Reference

Express Pcells

Sample Log File Format

Summary of Options:

dest	expressPcells_merged1
src	expressPcells_pmos expressPcells_nmos
log	xpcMerge1.log
noDestMerge	

VLS L license has been checked out for the Express Pcell feature.

VLS L license has been locked for Express Pcell feature. This will remain checked out until the current session ends.

Super Masters(lib/cell/view) in cache path, expressPcells_pmos :
gpdK090/pmos1v/layout 2

Super Masters(lib/cell/view) in cache path, expressPcells_nmos :
gpdK090/nmos1v/layout 1

Super Masters(lib/cell/view) and corresponding number of subMasters in merged cache, expressPcells_merged1 :

gpdK090/pmos1v/layout	2
gpdK090/nmos1v/layout	1

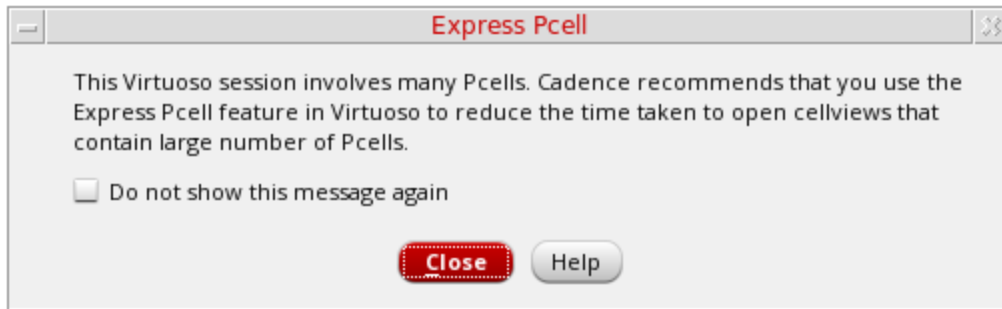
xpcmerge completed successfully. '0' errors and '0' warnings.

Virtuoso Parameterized Cell Reference

Express Pcells

Environment Variable to Control Express Pcell Proliferation Message

When Express Pcell is not enabled and your design activity involves a large number of Pcells, you will see the following Express Pcell Proliferation message.



Selecting the *Do not show this message again* check box sets the value of the `disableProliferationMsg` environment variable to `t` in the `.cdsenv` file and prevents this message box from being displayed during further Virtuoso sessions.

You can also define the following environment variable in the `.cdsenv` file:

```
xpcell disableProliferationMsg boolean { t | nil }
```

This environment variable lets you specify whether this message box should be displayed. The default is `nil`, which means that the message box is displayed.

Note: It is recommended to set this environment variable in `.cdsenv` file only. You cannot use the `envSetVal` command in CIW to change the value of this environment variable.

Exclude Specified Pcells from the Express Pcell Save or Read Operation

You can use the following methods to exclude specified Pcells from the Express Pcell cache during the save or read operation:

■ Specify Property String

Add a string property in the Pcell master as part of the Pcell code.

Example

```
procedure( myPcellCreate( libName cellName viewName)
pcDefinePCell(
    list(ddGetObj(libName) cellName viewName)
    (
        (length "int" 6)
    )
    let((cv)
        cv = pcCellView
        dbCreateRect(cv list("M1" "drawing") list(0:0 length:2))
        ; Add "xPcellExcluded" string property in layout/layout.oa
        dbReplaceProp(cv "xPcellExcluded" "string" "This Pcell is excluded
from Express Pcell operation.")
    ) ;let
);
)
```

Here, `xPcellExcluded` is a property of type `string`. It is used to exclude the specified Pcells from cache operations. The value of this string property does not matter.

■ Define cell exclusion SKILL list

Define the cell exclusion SKILL variable `xPcell_Excluded_libName` which is used to specify the list of Pcells in the `libInit.il` of the respective library.

Here `libName` is the name of the library for which cells must be excluded.

Lets consider a library `testLib`. You can exclude cells or views by defining the list in the following formats.

□ Specify Cells

Define a list of cell names to be excluded. When you use this method, all cellviews corresponding to the specified cells are excluded from the Express Pcell operation.

```
xPcell_Excluded_testLib = '(t_cellNames)
```

Virtuoso Parameterized Cell Reference

Express Pcells

For example, `xPcell_Excluded_testLib = '("cell1" "cell2" "cells3")`

In this example, all cellviews corresponding to the cells `cell1`, `cell2`, and `cell2` are excluded from `testLib/cell1`, `testLib/cell2` and `testLib/cell3`.

❑ Specify Cellviews

Define a list of cells to exclude all its cellviews. Additionally, define a list of cellviews for the specific cells to be excluded.

```
xPcell_Excluded_testLib = '('(t_cellNames) | '(t_cellNames  
t_viewNames))
```

For example,

```
xPcell_Excluded_testLib = '("cell1") ("cell2" "layout1") ("cell2"  
"layout2") ("cell3" "layout1"))
```

In this example, following cellviews will be excluded:

- All cellviews in `testLib/cell1`.
- Cellviews `testLib/cell2/layout1`, `testLib/cell2/layout2`, and `testLib/cell3/layout1`.

❑ Specify cellviews to be excluded for the specified cells

Define a list of cell names to exclude all related cellviews. Additionally, specify the specific views that should be excluded for the specified cells.

```
xPcell_Excluded_testLib = '('(t_cellNames) | '(t_cellNames  
'(t_viewNames))
```

For example,

```
xPcell_Excluded_testLib = '("cell1") ("cell2" ("layout1" "layout2"))  
("cell3" ("layout1"))
```

In this example, following cellviews will be excluded:

- All cellview of `testLib/cell1`
- Cellviews `testLib/cell2/layout1` and `testLib/cell2/layout2`
- Cellview `testLib/cell3/layout1`

You can use any of the list formats as per your requirement.

While specifying Pcell masters to be excluded during save and read operations, you can use both property string and cell exclusion SKILL list methods. In this case, Pcells masters specified by both methods are excluded.

Virtuoso Parameterized Cell Reference

Express Pcells

Graphical Pcells Form Descriptions

[Compile To Pcell Form](#) on page 251

[Compile To Skill Form](#) on page 252

[Conditional Inclusion Form](#) on page 253

[Define Parameterized Label Form](#) on page 254

[Define Parameterized Layer Form](#) on page 256

[Define Parameterized Path Form](#) on page 258

[Define Parameterized Polygon Form](#) on page 259

[Define Parameterized Rectangle Form](#) on page 260

[Define/Modify Inherited Parameters Form](#) on page 261

[Delete Conditional Inclusion Form](#) on page 262

[Delete Parameterized Layer Form](#) on page 263

[Delete Parameterized Path Form](#) on page 264

[Delete Parameterized Polygon Form](#) on page 265

[Delete Parameterized Property Form](#) on page 266

[Delete Parameterized Rectangle Form](#) on page 267

[Delete Reference Point Form](#) on page 268

[Delete Reference Point by Path Form](#) on page 269

[Edit Parameters Form](#) on page 270

[Modify Conditional Inclusion Form](#) on page 271

Virtuoso Parameterized Cell Reference

Graphical Pcells Form Descriptions

[Modify Parameterized Label Form](#) on page 272

[Modify Parameterized Layer Form](#) on page 274

[Modify Repeat in X Form](#) on page 275

[Modify Repeat in X and Y Form](#) on page 276

[Modify Repeat in Y Form](#) on page 278

[Modify Repetition Along Shape Form](#) on page 279

[Parameterized Property Form](#) on page 280

[Reference Point by Path Endpoint Form](#) on page 281

[Reference Point by Parameter Form](#) on page 282

[Repeat in X Form](#) on page 283

[Repeat in X and Y Form](#) on page 284

[Repeat in Y Form](#) on page 286

[Repetition Along Shape Form](#) on page 287

[Stretch in X Form](#) on page 288

[Stretch in Y Form](#) on page 290

[Ultra Pcell Form](#) on page 292

Compile To Pcell Form

See [Creating a Pcell from a Cellview](#) for more information.

Function is the name for the function property assigned to the Pcell.

Valid Values: *transistor*, *contact*, *substrateContact*, *none*

Default: *transistor*

Compile To Skill Form

See [Creating a SKILL File from a Cellview](#) for more information.

Library Name is the name of the library where the cellview is located.

Cell Name is the name you assign to the cellview as it is defined in the SKILL file. Give it a name that is different from the layout cellview name. When you load the SKILL file, this new name is the one used for the new layout cell created.

Valid Values: any valid cellview name

View Name is the view name of the current cellview.

Valid Values: *layout*, *symbol*, *schematic*, and any others defined in the technology file

File Name is the path for the SKILL file. When you give the path for the SKILL file, do not put the file in the library where the cellview is located. This type of library is reserved for files created by the layout editor.

Valid Values: any directory other than a layout editor library

Conditional Inclusion Form

See [Conditional Inclusion Commands](#) for more information.

Name or Expression is the definition of the parameter or SKILL expression. When you place an instance of the Pcell, the Pcell program evaluates this expression to determine whether to include the specified objects.

A parameter name must

- Be a single word starting with a letter
- Contain no special characters
- Be no more than 32 characters in length

Valid Values: any parameter definition or SKILL expression that evaluates to `t` or `nil`

Dependent Stretch is the name of a previously defined stretch control line. You can set this stretch control line to be dependent on conditionally included objects.

Valid Values: name of any stretch control line

Adjustment to Stretch is the amount the stretch control line stretches or compresses when the conditional objects are not included. Adjustment to stretch can be positive or negative.

Valid Values: integer or floating-point number

Default: 0.0

Define Parameterized Label Form

See [Parameterized Label Commands](#) for more information.

Label is the definition of the text you want displayed.

Valid Values: any SKILL expression

Height is the height of the text in user units (usually microns), or a SKILL expression.

Valid Values: any number greater than zero or a valid SKILL expression

Font is the text font style you want to use.

Valid Values: *europe*, *gothic*, *roman*, *script*, *stick*

europe roman stick
gothic script

Transform controls the orientation of the text. The default is 0 (no rotation), which is standard text orientation, left to right.

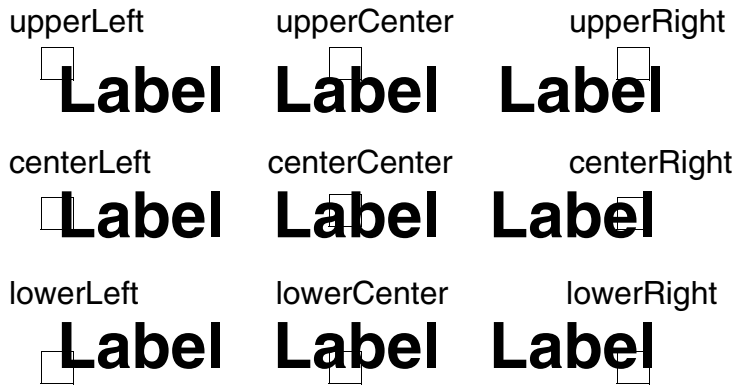
Valid Values:

<i>0</i>	No rotation
<i>90</i>	Rotate 90 degrees
<i>180</i>	Rotate 180
<i>270</i>	Rotate 270
<i>mirror x</i>	Mirror over the X axis
<i>mirror y</i>	Mirror over the Y axis
<i>mirror x rotate 90</i>	Mirror over the X axis and rotate 90 degrees
<i>mirror x rotate 270</i>	Mirror over the X axis and rotate 270 degrees

Virtuoso Parameterized Cell Reference

Graphical Pcells Form Descriptions

Justification controls the origin of the text. For example, *upperLeft* sets the origin to the upper left corner of the text. When you enter a coordinate to place the text, the origin of the text appears at that coordinate.



Drafting determines whether you can rotate text in all directions: when turned on, it prevents text from rotating more than 90 degrees, so it remains readable; when turned off, it lets you rotate text in all directions using the *Transform* button, including mirrored rotations that might be difficult to read.

Define Parameterized Layer Form

See [Parameterized Layer Commands](#) for more information.

Layer Parameter or Expression is the definition for the parameter that controls the layer on which the selected objects are drawn.

Valid Values: any SKILL expression that evaluates to a string

Purpose Parameter or Expression is the definition for the parameter that controls the purpose of the layer on which the selected objects are drawn. You must define a layer name parameter before you can define a purpose parameter.

Valid Values: any SKILL expression that evaluates to a string

Layer Mask Value or Expression specifies the mask color of the shapes in the parameterized layer. (ICADVM18.1 only)

a. Example of layer mask parameter with string data type:

- “mask1color”: Shape is assigned to mask1
- “mask2color”: Shape is assigned to mask2

b. Example of layer mask parameter with integer data type:

- 1: Shape is assigned to mask1
- 2: Shape is assigned to mask2

c. Example of expression in layer mask parameter:

```
mod(maskNum+numTracks 2) + 1
```

where, maskNum and numTracks are existing integer Pcell parameters.

Lock State Value or Expression specifies whether or not the shape’s mask color is locked. (ICADVM18.1 only)

a. Example of lock state parameter with string data type:

- “t”: Shape is locked
- “nil”: Shape is unlocked
- “locked”: Shape is unlocked
- “foo”: Shape is unlocked

Note: The value “t” indicates the locked state and other string values indicate the

Virtuoso Parameterized Cell Reference

Graphical Pcells Form Descriptions

unlocked state.

b. Example of lock state parameter with Boolean data type:

- ☐ `t`: Shape is locked
- ☐ `nil`: Shape is unlocked

Define Parameterized Path Form

See [Parameterized Shapes Commands](#) for more information.

Parameter Name is the definition of the parameter controlling the vertexes of the path. The default name is `coords` and cannot be changed. Although you can have only one parameter name, you can associate multiple parameterized paths with a single parameter name by using this command more than once.

Valid Values: a SKILL symbol

Snap controls how line segments are drawn between points you enter.

orthogonal draws straight line segments connected at 90 degrees at each point.

diagonal draws straight line segments parallel to the X axis, Y axis, or at a 45-degree angle to the axes.

anyAngle draws straight lines at any angle.

Margin is the amount to offset the centerline of the path from the digitized coordinates.

Margin can be used only when *Snap* is set to *orthogonal*.

Valid Values: a positive or negative number or any SKILL expression that evaluates to a positive or negative number

Width is the parameterized path width.

Valid Values: any SKILL expression

Define Parameterized Polygon Form

See [Parameterized Shapes Commands](#) for more information.

Parameter Name is the definition of the parameter controlling the vertexes of the path. The default name is `coords` and cannot be changed. Although you can have only one parameter name, you can associate multiple parameterized paths with a single parameter name by using this command more than once.

Valid Values: a SKILL symbol

Snap controls how line segments are drawn between points you enter.

orthogonal draws straight line segments connected at 90 degrees at each point.

diagonal draws straight line segments parallel to the X axis, Y axis, or at a 45-degree angle to the axes.

anyAngle draws straight lines at any angle.

Margin is the amount to offset the centerline of the path from the digitized coordinates.

Margin can be used only when *Snap* is set to *orthogonal*.

Valid Values: a positive or negative number or any SKILL expression that evaluates to a positive or negative number

Width is the parameterized path width.

Valid Values: any SKILL expression

Define Parameterized Rectangle Form

See [Parameterized Shapes Commands](#) for more information.

Parameter Name is the definition of the parameter controlling the vertexes of the path. The default name is `coords` and cannot be changed. Although you can have only one parameter name, you can associate multiple parameterized paths with a single parameter name by using this command more than once.

Valid Values: a SKILL symbol

Snap controls how line segments are drawn between points you enter.

orthogonal draws straight line segments connected at 90 degrees at each point.

diagonal draws straight line segments parallel to the X axis, Y axis, or at a 45-degree angle to the axes.

anyAngle draws straight lines at any angle.

Margin is the amount to offset the centerline of the path from the digitized coordinates.

Margin can be used only when *Snap* is set to *orthogonal*.

Valid Values: a positive or negative number or any SKILL expression that evaluates to a positive or negative number

Width is the parameterized path width.

Valid Values: any SKILL expression

Define/Modify Inherited Parameters Form

See [Inherited Parameters Commands](#) for more information.

The form lists each parameter and its current value. For each parameter, there is an *inherit* button. If you turn this button off, the parameter name and field are shaded and you cannot change the parameter value. If you turn this button on, the parameter name is bold and you can change the parameter value.

Valid Values: a name or any valid SKILL expression.

Delete Conditional Inclusion Form

See [Including or Excluding Objects](#) for more information.

Name or Expression is the definition of the parameter or SKILL expression. When you place an instance of the Pcell, the Pcell program evaluates this expression to determine whether to include the specified objects.

A parameter name must

- Be a single word starting with a letter
- Contain no special characters
- Be no more than 32 characters in length

Valid Values: any parameter definition or SKILL expression that evaluates to `t` or `nil`

Dependent Stretch is the name of a previously defined stretch control line. You can set this stretch control line to be dependent on conditionally included objects.

Valid Values: name of any stretch control line

Adjustment to Stretch is the amount the stretch control line stretches or compresses when the conditional objects are not included. Adjustment to stretch can be positive or negative.

Valid Values: integer or floating-point number

Default: 0.0

Delete Parameterized Layer Form

See [Parameterized Label Commands](#) for more information.

Layer Parameter or Expression is the definition for the parameter that controls the layer on which the selected objects are drawn.

Valid Values: any SKILL expression that evaluates to a string

Purpose Parameter or Expression is the definition for the parameter that controls the purpose of the layer on which the selected objects are drawn. You must define a layer name parameter before you can define a purpose parameter.

Valid Values: any SKILL expression that evaluates to a string

Delete Parameterized Path Form

See [Parameterized Shapes Commands](#) for more information.

Parameter Name is the definition of the parameter controlling the vertexes of the path. The default name is `coords` and cannot be changed. Although you can have only one parameter name, you can associate multiple parameterized paths with a single parameter name by using this command more than once.

Valid Values: a SKILL symbol

Snap controls how line segments are drawn between points you enter.

orthogonal draws straight line segments connected at 90 degrees at each point.

diagonal draws straight line segments parallel to the X axis, Y axis, or at a 45-degree angle to the axes.

anyAngle draws straight lines at any angle.

Margin is the amount to offset the centerline of the path from the digitized coordinates.

Margin can be used only when *Snap* is set to *orthogonal*.

Valid Values: a positive or negative number or any SKILL expression that evaluates to a positive or negative number

Width is the parameterized path width.

Valid Values: any SKILL expression

Delete Parameterized Polygon Form

See [Parameterized Shapes Commands](#) for more information.

Parameter Name is the definition of the parameter controlling the vertexes of the path. The default name is `coords` and cannot be changed. Although you can have only one parameter name, you can associate multiple parameterized paths with a single parameter name by using this command more than once.

Valid Values: a SKILL symbol

Snap controls how line segments are drawn between points you enter.

orthogonal draws straight line segments connected at 90 degrees at each point.

diagonal draws straight line segments parallel to the X axis, Y axis, or at a 45-degree angle to the axes.

anyAngle draws straight lines at any angle.

Margin is the amount to offset the centerline of the path from the digitized coordinates.

Margin can be used only when *Snap* is set to *orthogonal*.

Valid Values: a positive or negative number or any SKILL expression that evaluates to a positive or negative number

Width is the parameterized path width.

Valid Values: any SKILL expression

Delete Parameterized Property Form

See [Parameterized Property Commands](#) for more information.

Property Name is a property name you define.

Valid Values: any SKILL expression that evaluates to a string

Name or Expression for Property is a name or expression for the value of the property being defined.

Valid Values: any SKILL expression that evaluates to a string

Delete Parameterized Rectangle Form

See [Parameterized Shapes Commands](#) for more information.

Parameter Name is the definition of the parameter controlling the vertexes of the path. The default name is `coords` and cannot be changed. Although you can have only one parameter name, you can associate multiple parameterized paths with a single parameter name by using this command more than once.

Valid Values: a SKILL symbol

Snap controls how line segments are drawn between points you enter.

orthogonal draws straight line segments connected at 90 degrees at each point.

diagonal draws straight line segments parallel to the X axis, Y axis, or at a 45-degree angle to the axes.

anyAngle draws straight lines at any angle.

Margin is the amount to offset the centerline of the path from the digitized coordinates.

Margin can be used only when *Snap* is set to *orthogonal*.

Valid Values: a positive or negative number or any SKILL expression that evaluates to a positive or negative number

Width is the parameterized path width.

Valid Values: any SKILL expression

Delete Reference Point Form

See [Reference Point Commands](#) for more information.

Parameter Name is the parameter name assigned to the reference point.
Valid Values: any SKILL symbol

Delete Reference Point by Path Form

See [Reference Point Commands](#) for more information.

Path Parameter Name is the name of the parameter controlling the vertexes of the path. The default name is `coords` and cannot be changed. Although you can have only one parameter name in a single cellview, you can associate multiple parameterized paths with a single parameter name by using this command more than once.

Endpoint of the Path is the end of the path you want to use as the reference point.

first indicates the beginning of the path (the first point entered).

last indicates the end of the path (the last point entered).

Edit Parameters Form

See [Parameters Commands](#) for more information.

Parameter Name is the name you assigned to each parameter in the current cellview. You cannot change this field.

Data Type is the data type of each parameter.

Valid Values: *Integer, Float, String, Boolean, ILList*

Value is the default value the system uses for the parameter when you place an instance.

Valid Values: any value consistent with the data type specified in the *Data Type* field

Reset to Defaults resets the *Data Type* and *Value* fields for a parameter back to the value defined for the parameter when the parameter was added to the master Pcell.

Modify Conditional Inclusion Form

See [Conditional Inclusion Commands](#) for more information.

Name or Expression is the definition of the parameter or SKILL expression. When you place an instance of the Pcell, the Pcell program evaluates this expression to determine whether to include the specified objects.

A parameter name must

- Be a single word starting with a letter
- Contain no special characters
- Be no more than 32 characters in length

Valid Values: any parameter definition or SKILL expression that evaluates to `t` or `nil`

Dependent Stretch is the name of a previously defined stretch control line. You can set this stretch control line to be dependent on conditionally included objects.

Valid Values: name of any stretch control line

Adjustment to Stretch is the amount the stretch control line stretches or compresses when the conditional objects are not included. Adjustment to stretch can be positive or negative.

Valid Values: integer or floating-point number

Default: 0.0

Modify Parameterized Label Form

See [Parameterized Label Commands](#) for more information.

Label is the definition of the text you want displayed.

Valid Values: any SKILL expression

Height is the height of the text in user units (usually microns), or a SKILL expression.

Valid Values: any number greater than zero or a valid SKILL expression

Font is the text font style you want to use.

Valid Values: *europe*, *gothic*, *roman*, *script*, *stick*

europe roman stick
gothic script

Transform controls the orientation of the text. The default is 0 (no rotation), which is standard text orientation, left to right.

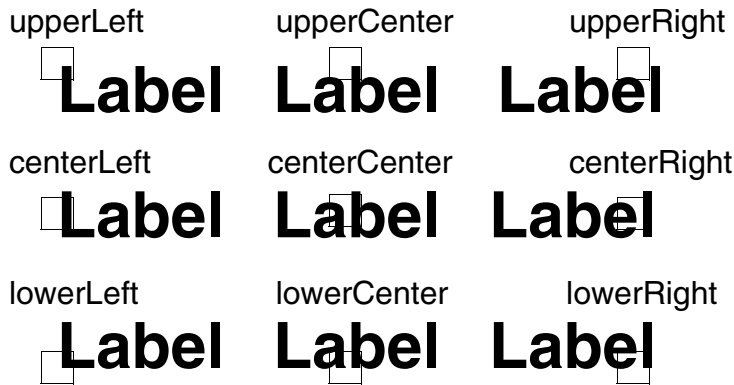
Valid Values:

<i>0</i>	No rotation
<i>90</i>	Rotate 90 degrees
<i>180</i>	Rotate 180
<i>270</i>	Rotate 270
<i>mirror x</i>	Mirror over the X axis
<i>mirror y</i>	Mirror over the Y axis
<i>mirror x rotate 90</i>	Mirror over the X axis and rotate 90 degrees
<i>mirror x rotate 270</i>	Mirror over the X axis and rotate 270 degrees

Virtuoso Parameterized Cell Reference

Graphical Pcells Form Descriptions

Justification controls the origin of the text. For example, *upperLeft* sets the origin to the upper left corner of the text. When you enter a coordinate to place the text, the origin of the text appears at that coordinate.



Drafting determines whether you can rotate text in all directions: when turned on, it prevents text from rotating more than 90 degrees, so it remains readable; when turned off, it lets you rotate text in all directions using the *Transform* button, including mirrored rotations that might be difficult to read.

Modify Parameterized Layer Form

See [Parameterized Layer Commands](#) for more information.

Layer Parameter or Expression is the definition for the parameter that controls the layer on which the selected objects are drawn.

Valid Values: any SKILL expression that evaluates to a string

Purpose Parameter or Expression is the definition for the parameter that controls the purpose of the layer on which the selected objects are drawn. You must define a layer name parameter before you can define a purpose parameter.

Valid Values: any SKILL expression that evaluates to a string

Layer Mask Value or Expression specifies the mask color of the shapes in the parameterized layer. (ICADVM18.1 only)

Lock State Value or Expression specifies whether or not the shape's mask color is locked. (ICADVM18.1 only)

For more information, see [Define Parameterized Layer Form](#).

Modify Repeat in X Form

See [Repetition Along Shape Commands](#) for more information.

Stepping Distance is the centerline-to-centerline distance (pitch) you want between repeated objects. To step objects to the left, use a negative number, such as `-(length + 4)`.

You can use a SKILL expression that is dependent on other parameters in the Pcell, such as `length + 4` or an expression that uses technology-file parameters, such as

```
techGetSpacingRule ( techGetTechFile (pcCellView)
                    "minWidth"
                    list ("via" "drawing")
)
+
techGetSpacingRule (
                    techGetTechFile (pcCellView)
                    "minSpacing"
                    list ("via" "drawing")
)
```

Valid Values: any SKILL expression

Number of Repetitions is the number of times you want the specified objects repeated. You can use a number or an expression that is dependent on other parameters of the Pcell, such as `length`.

Valid Values: any SKILL expression

Dependent Stretch is the name of a previously defined stretch control line. If you specify a dependent stretch control line, stretching takes place after repetition.

Valid Values: any stretch control line name

Adjustment to Stretch is the amount the program adjusts the reference dimension of the stretch parameter. The expression for *Adjustment to Stretch* usually uses the values in *Stepping Distance* and *Number of Repetitions*.

Valid Values: any SKILL expression

Default: `(fix(pcRepeatX) -1) * pcStepX`

Modify Repeat in X and Y Form

See [Repetition Along Shape Commands](#) for more information.

X Stepping Distance is the horizontal centerline-to-centerline distance (pitch) you want between repeated objects.

You can use a SKILL expression that is dependent on other parameters in the Pcell, such as

`width+4`

or an expression that uses technology-file parameters, such as

```
pcTechFile(  
    techGetSpacingRule(  
        techGetTechFile(pcCellView)  
        "min Width" list("via" "drawing")  
    )  
    +  
    techGetSpacingRule(  
        techGetTechFile(pcCellView)  
        "min Spacing" list("via" "drawing")  
    )  
)
```

Valid Values: any SKILL expression

Y Stepping Distance is the vertical centerline-to-centerline distance (pitch) you want between repeated objects. You can use a number or an expression that is dependent on other parameters in the Pcell or an expression that uses technology file parameters.

Valid Values: any SKILL expression

Number of X Repetitions is the number of times you want the objects repeated horizontally. You can use a number or an expression that is dependent on other parameters; for example, the number of repetitions for contacts in a multigate transistor might be `gates+1`.

Valid Values: any SKILL expression

Number of Y Repetitions is the number of times you want the objects repeated vertically. You can use a number or an expression that is dependent on other parameters, such as the width of the gates.

Valid Values: any SKILL expression

Dependent X Stretch is the name of a previously defined horizontal stretch control line. If you specify a dependent stretch control line, stretching takes place after repetition.

Valid Values: any stretch control line name

Dependent Y Stretch is the name of a previously defined vertical stretch control line. If you specify a dependent stretch control line, stretching takes place after repetition.

Valid Values: any stretch control line name

Virtuoso Parameterized Cell Reference

Graphical Pcells Form Descriptions

Adjustment to X Stretch is the amount added to or subtracted from the reference dimension of the stretch parameter. This expression usually uses the values in *Stepping Distance* and *Number of Repetitions*.

Valid Values: any SKILL expression

Default: `(fix(pcRepeatX) -1) * pcStepX)`

Adjustment to Y Stretch is the amount added or subtracted from the reference dimension of the stretch parameter. This expression usually uses the values in *Stepping Distance* and *Number of Repetitions*.

Valid Values: any SKILL expression

Default: `(fix(pcRepeatY) -1) * pcStepY)`

Modify Repeat in Y Form

See [Repetition Along Shape Commands](#) for more information.

Stepping Distance is the centerline-to-centerline distance (pitch) you want between repeated objects. To step objects downward, use a negative number.

You can use a SKILL expression that is dependent on other parameters in the Pcell, such as `width/6` or an expression that uses technology-file parameters, such as

```
pcTechFile (
    techGetSpacingRule (
        techGetTechFile (pcCellView)
        min Width" list("via" "drawing")
    )
    + techGetSpacingRule (
        techGetTechFile (pcCellView)
        "min Spacing" list("via" "drawing")
    )
)
```

Valid Values: any SKILL expression

Number of Repetitions is the number of times you want the objects repeated. You can use a number or an expression that is dependent on other parameters of the Pcell, such as `width`.

Valid Values: any SKILL expression

Dependent Stretch is the name of a previously defined stretch control line. If you specify a dependent stretch control line, stretching takes place after repetition.

Valid Values: any stretch control line name

Adjustment to Stretch is the amount the program adds or subtracts from the reference dimension of the stretch parameter. This expression usually uses the values in *Stepping Distance* and *Number of Repetitions*.

Valid Values: any SKILL expression

Default: `(fix(pcRepeatY) -1) * pcStepY`

Modify Repetition Along Shape Form

See [Repetition Along Shape Commands](#) for more information.

Parameter Name is the name of the parameter controlling the vertexes of the shape. The default name is `coords` and cannot be changed. Although you can have only one parameter name, you can associate multiple parameterized shapes with a single parameter name by using this command more than once.

Stepping Distance is the centerline-to-centerline distance (pitch) you want between the repeated objects.

Valid Values: any SKILL expression

Start Offset is the gap the Pcell program leaves from the first vertex of the shape to the first repeated object.

Valid Values: any SKILL expression

End Offset is the gap the Pcell program leaves from the last repeated object to the last entered vertex of the shape.

Valid Values: any SKILL expression

Parameterized Property Form

See [Parameterized Property Commands](#) for more information.

Property Name is a property name you define.

Valid Values: any SKILL expression that evaluates to a string

Name or Expression for Property is a name or expression for the value of the property being defined.

Valid Values: any SKILL expression that evaluates to a string

Reference Point by Path Endpoint Form

See [Reference Point Commands](#) for more information.

Path Parameter Name is the name of the parameter controlling the vertexes of the path. The default name is `coords` and cannot be changed. Although you can have only one parameter name in a single cellview, you can associate multiple parameterized paths with a single parameter name by using this command more than once.

Endpoint of the Path is the end of the path you want to use as the reference point.

first indicates the beginning of the path (the first point entered).

last indicates the end of the path (the last point entered).

Reference Point by Parameter Form

See [Reference Point Commands](#) for more information.

Parameter Name is the parameter name assigned to the reference point.
Valid Values: any SKILL symbol

Repeat in X Form

See [Repetition Commands](#) for more information.

Stepping Distance is the centerline-to-centerline distance (pitch) you want between repeated objects. To step objects to the left, use a negative number, such as `-(length + 4)`.

You can use a SKILL expression that is dependent on other parameters in the Pcell, such as `length + 4` or an expression that uses technology-file parameters, such as

```
techGetSpacingRule ( techGetTechFile (pcCellView)
                    "minWidth"
                    list ("via" "drawing")
                    )
+
techGetSpacingRule ( techGetTechFile (pcCellView)
                    "minSpacing"
                    list ("via" "drawing")
                    )
```

Valid Values: any SKILL expression

Number of Repetitions is the number of times you want the specified objects repeated. You can use a number or an expression that is dependent on other parameters of the Pcell, such as `length`.

Valid Values: any SKILL expression

Dependent Stretch is the name of a previously defined stretch control line. If you specify a dependent stretch control line, stretching takes place after repetition.

Valid Values: any stretch control line name

Adjustment to Stretch is the amount the program adjusts the reference dimension of the stretch parameter. The expression for *Adjustment to Stretch* usually uses the values in *Stepping Distance* and *Number of Repetitions*.

Valid Values: any SKILL expression

Default: `(fix(pcRepeatX) -1) * pcStepX`

Repeat in X and Y Form

See [Repetition Commands](#) for more information.

X Stepping Distance is the horizontal centerline-to-centerline distance (pitch) you want between repeated objects.

You can use a SKILL expression that is dependent on other parameters in the Pcell, such as

`width+4`

or an expression that uses technology-file parameters, such as

```
pcTechFile(  
    techGetSpacingRule(  
        techGetTechFile(pcCellView)  
        "min Width" list("via" "drawing")  
    )  
    +  
    techGetSpacingRule(  
        techGetTechFile(pcCellView)  
        "min Spacing" list("via" "drawing")  
    )  
)
```

Valid Values: any SKILL expression

Y Stepping Distance is the vertical centerline-to-centerline distance (pitch) you want between repeated objects. You can use a number or an expression that is dependent on other parameters in the Pcell or an expression that uses technology file parameters.

Valid Values: any SKILL expression

Number of X Repetitions is the number of times you want the objects repeated horizontally. You can use a number or an expression that is dependent on other parameters; for example, the number of repetitions for contacts in a multigate transistor might be `gates+1`.

Valid Values: any SKILL expression

Number of Y Repetitions is the number of times you want the objects repeated vertically. You can use a number or an expression that is dependent on other parameters, such as the width of the gates.

Valid Values: any SKILL expression

Dependent X Stretch is the name of a previously defined horizontal stretch control line. If you specify a dependent stretch control line, stretching takes place after repetition.

Valid Values: any stretch control line name

Dependent Y Stretch is the name of a previously defined vertical stretch control line. If you specify a dependent stretch control line, stretching takes place after repetition.

Valid Values: any stretch control line name

Virtuoso Parameterized Cell Reference

Graphical Pcells Form Descriptions

Adjustment to X Stretch is the amount added to or subtracted from the reference dimension of the stretch parameter. This expression usually uses the values in *Stepping Distance* and *Number of Repetitions*.

Valid Values: any SKILL expression

Default: `(fix(pcRepeatX) -1) * pcStepX)`

Adjustment to Y Stretch is the amount added or subtracted from the reference dimension of the stretch parameter. This expression usually uses the values in *Stepping Distance* and *Number of Repetitions*.

Valid Values: any SKILL expression

Default: `(fix(pcRepeatY) -1) * pcStepY)`

Repeat in Y Form

See [Repetition Commands](#) for more information.

Stepping Distance is the centerline-to-centerline distance (pitch) you want between repeated objects. To step objects downward, use a negative number.

You can use a SKILL expression that is dependent on other parameters in the Pcell, such as `width/6` or an expression that uses technology-file parameters, such as

```
pcTechFile (
    techGetSpacingRule (
        techGetTechFile (pcCellView)
        min Width" list("via" "drawing")
    )
    + techGetSpacingRule (
        techGetTechFile (pcCellView)
        "min Spacing" list("via" "drawing")
    )
)
```

Valid Values: any SKILL expression

Number of Repetitions is the number of times you want the objects repeated. You can use a number or an expression that is dependent on other parameters of the Pcell, such as `width`.

Valid Values: any SKILL expression

Dependent Stretch is the name of a previously defined stretch control line. If you specify a dependent stretch control line, stretching takes place after repetition.

Valid Values: any stretch control line name

Adjustment to Stretch is the amount the program adds or subtracts from the reference dimension of the stretch parameter. This expression usually uses the values in *Stepping Distance* and *Number of Repetitions*.

Valid Values: any SKILL expression

Default: `(fix(pcRepeatY) -1) * pcStepY`

Repetition Along Shape Form

See [Repetition Along Shape Commands](#) for more information.

Parameter Name is the name of the parameter controlling the vertexes of the shape. The default name is `coords` and cannot be changed. Although you can have only one parameter name, you can associate multiple parameterized shapes with a single parameter name by using this command more than once.

Stepping Distance is the centerline-to-centerline distance (pitch) you want between the repeated objects.

Valid Values: any SKILL expression

Start Offset is the gap the Pcell program leaves from the first vertex of the shape to the first repeated object.

Valid Values: any SKILL expression

End Offset is the gap the Pcell program leaves from the last repeated object to the last entered vertex of the shape.

Valid Values: any SKILL expression

Stretch in X Form

See [Stretch Commands](#) for more information.

Name or Expression for Stretch is a parameter name or SKILL expression. A parameter name, such as `length`, must

- Be a single word starting with a letter
- Contain no special characters
- Be no more than 32 characters in length

When you specify an expression, the environment variable `stretchPCellApplyToName` controls how the system uses the *Reference Dimension (default)* field and the *Minimum* and *Maximum* fields. The `stretchPCellApplyToName` environment variable is set to `t` by default. For more information about this variable, see [Applying Default, Minimum, and Maximum Values](#).

Valid Values: any name or SKILL expression

Reference Dimension (Default) is the default value for the stretch control line. You typically use a measurement from an object in the cell, such as the length or width of a gate. When you place a Pcell, the system subtracts the [reference dimension](#) from the stretch value you specify to determine the distance to stretch.

Valid Values: integer or floating-point number

Default: length of the shortest edge crossed by the stretch control line

Stretch Direction is the direction you want objects in the cellview to stretch.

right stretches objects to the right.

left stretches objects to the left.

right and left stretches objects in each direction half the total stretch value.

Stretch Horizontally Repeated Figures stretches objects marked for repetition in the X direction. By default, objects in repetition groups are not stretched.

Minimum Value specifies the smallest value you can use to stretch this Pcell. If you specify a value smaller than the minimum, the Pcell program uses the previous value.

Valid Values: integer or floating-point number

Default: 0.0 (no range checking)

Maximum Value specifies the largest value you can use to stretch this Pcell. If you specify a value greater than the maximum, the Pcell program uses the previous value.

Virtuoso Parameterized Cell Reference

Graphical Pcells Form Descriptions

Valid Values: integer or floating-point number

Default: 0.0 (no range checking)

Stretch in Y Form

See [Stretch Commands](#) for more information.

Name or Expression for Stretch is a parameter name or SKILL expression. A parameter name, such as `length`, must

- Be a single word starting with a letter
- Contain no special characters
- Be no more than 32 characters in length

When you specify an expression, the environment variable `stretchPCellApplyToName` controls how the system uses the *Reference Dimension (default)* field and the *Minimum* and *Maximum* fields. The `stretchPCellApplyToName` environment variable is set to `t` by default. For more information about this variable, see [Applying Default, Minimum, and Maximum Values](#).

Valid Values: any name or SKILL expression

Reference Dimension (Default) is the default value for the stretch control line. You typically use a measurement from an object in the cell, such as the length or width of a gate. When you place a Pcell, the system subtracts the [reference dimension](#) from the stretch value you specify to determine the distance to stretch.

Valid Values: integer or floating-point number

Default: length of the shortest edge crossed by the stretch control line

Stretch Direction is the direction you want objects in the cellview to stretch.

up stretches objects upward.

down stretches objects downward.

up and down stretches objects in each direction half the total stretch value.

Stretch Vertically Repeated Figures stretches objects marked for repetition in the Y direction. By default, objects in repetition groups are not stretched.

Minimum Value specifies the smallest value you can use to stretch this Pcell. If you specify a value smaller than the minimum, the Pcell program uses the previous value.

Valid Values: integer or floating-point number

Default: 0.0 (no range checking)

Maximum Value specifies the largest value you can use to stretch this Pcell. If you specify a value greater than the maximum, the Pcell program uses the previous value.

Virtuoso Parameterized Cell Reference

Graphical Pcells Form Descriptions

Valid Values: integer or floating-point number

Default: 0.0 (no range checking)

Virtuoso Parameterized Cell Reference

Graphical Pcells Form Descriptions

Ultra Pcell Form

See [Make Ultra Pcell Command](#) for more information.

Selector Parameter is the name of the parameter used to select the layout.

There are illegal parameter names such as *type*, *objType*, *name*, *cell*, and *status*. These are reserved as CDBA (Cadence database) query keywords. If you use one of these names, the ultra Pcell generator prompts you for another name.

Library, **Cell**, and **View** are the library, cell, and view names of the final compiled ultra Pcell.

of Cells displays the total number of cells making up this ultra Pcell.

Add Cell displays more of the ultra Pcell form that lets you add another cell.

Enter value of the selector parameter... →

...for this Pcell. →

The screenshot shows the 'Ultra Pcell' dialog box. At the top are buttons for 'OK', 'Cancel', 'Apply', and 'Help'. The main area contains the following fields and controls:

- Selector Parameter:** A text field containing the value 'a'. An arrow points to this field from the text 'Enter value of the selector parameter...'.
- Library:** A text field containing the value 'tutorial'.
- Cell:** A text field containing the value 'bmcp'.
- View:** A text field containing the value 'layout'.
- # of Cells:** A text field containing the value '2'. An arrow points to this field from the text '...for this Pcell.'.
- Add Cell:** A button located below the '# of Cells' field.
- Browse:** A button located to the right of the 'Add Cell' button.
- Selector Value Section 1:** A section containing a text field with the value '65.31', a 'Delete Cell' button, and three smaller text fields for 'Library' (tutorial), 'Cell' (bmcp), and 'View' (layout).
- Selector Value Section 2:** A section containing a text field with the value '8.16', a 'Delete Cell' button, and three smaller text fields for 'Library' (tutorial), 'Cell' (bmcp), and 'View' (layout).

Browse opens the Browser, which lets you select the cell and view.

Virtuoso Parameterized Cell Reference

Graphical Pcells Form Descriptions

Selector Value is the value of the selector parameter to be used to generate the desired layout.

Library displays the library of this cell.

Cell and **View** are the cell and view name of the Pcell.

Delete Cell removes that cell.

Virtuoso Parameterized Cell Reference

Graphical Pcells Form Descriptions

Express Pcell Manager

See [Using the Express Pcell Manager GUI in a Virtuoso Session](#) for more information.

Express Pcells directory specifies the directory where the cached Pcells are stored. By default, the cached Pcells are stored in the `.expressPcells` directory that is created in your current working directory.

Environment variable: [CDS_EXP_PCELL_DIR](#)

Enable Express Pcells controls the caching of evaluated SKILL Pcells. By default, the caching of Pcells is disabled.

Environment variable: [CDS_ENABLE_EXP_PCELL](#)

Auto Save allows the automatic update of Pcell cache on disk for any new submasters generated in virtual memory with every *File – Save* operation. This option is selected by default. If you deselect it, you are prompted with the following message each time you save a design containing a new Pcell variant in the memory:

```
You have some PCell data which is not saved to cache. Do you want to save it now?
```

Save Cache saves all the Pcell submasters in the virtual memory (after evaluation) to the disk.

Pop-Up Context Menu Items

These menu commands apply at the level of the selected context — a library or a cellview.

- **Ignore TimeStamp** ignores the timestamps of Pcell supermasters while reading or updating their corresponding variants on disk. Therefore, when reading cached data, variants are read from the cache while opening the design irrespective of their timestamp. Similarly, during writing of cached data, only the new variants are cached to the disk irrespective of their timestamp. The timestamp icon appears against the selected cellviews on which the *Ignore TimeStamp* command is applied. If you apply the command anywhere in the *Express Pcells* navigation frame, without selecting any cellview, the context is considered to be *All Libs*.
- **Update Cache** checks the timestamps of the cached supermasters and if any existing data in the Pcell cache is found to be out-of-date, it updates all the corresponding cached

Virtuoso Parameterized Cell Reference

Express Pcell Manager

submasters on disk as per their more recent supermasters. If, however, the *Ignore TimeStamp* option is selected for a supermaster in your current Virtuoso session, the *Update Cache* option does not update the cache on disk for that supermaster.

- **Clear Cache** deletes the saved variants of the selected library or cellview. If this command is applied at the top level on *All Libs*, all the cached data in the current cache directory is deleted.

Note: In multi-user shared cache environment, you must take care when using the *Clear Cache* feature, in case you accidentally delete Pcell data files from the cache that might be required by other users.