cadence®

Sample Parameterized Cells Installation and Reference

Product Version IC23.1
June 2023

Cadence Design Systems, Inc., 555 River Oaks Parkway, San Jose, CA 95134, USA

Contents

1

Overview and Installation

This manual contains information about installing and modifying a collection of sample parameterized cells (Pcells) built procedurally with relative object design (ROD) functions of the Cadence® SKILL programming language.

Using these sample Pcells requires knowledge of the Virtuoso layout editor and the SKILL programming language and its ROD functions. The following documents provide more information on these tools.

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

- The Virtuoso Studio 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 Studio design environment, notably, the Virtuoso Layout Suite, and Virtuoso Schematic Editor.
- The Virtuoso Studio design environment technology file.

Component description format (CDF), which lets you create and describe your own components for use with Layout XL.

This chapter contains conceptual information about the sample parameterized cells and a guide to installing these devices into your library.

About the Sample Parameterized Cells

Sample parameterized cells are layout views of devices built using relative object design (ROD) functions. A parameterized cell (Pcell) is a graphic, programmable cell that lets you

create a customized instance each time you place it. The Pcells in this collection serve as examples of how you can use ROD functions to create your own Pcells. You can modify the sample Pcells to suit your specifications. The files containing the device code reside in the directory

your_install_dir/tools/dfII/samples/ROD/rodPcells/components/*device*

where *your_install_dir* is the path to the Cadence® software directory and *device* is the directory containing the code for the device you want to modify. The following table provides the directory name for all of the sample Pcells.

Device Name	Directory
spcbentnmos/spcbentpmos	bentmos
spccap	cap
spcinv	inv
spcnmos/spcpmos	mos
spcnpn	npn
spcpnp	pnp
spcpres	pres
spcres	res
spcsimple_nmos/spcsimple_pmos	simple_mos

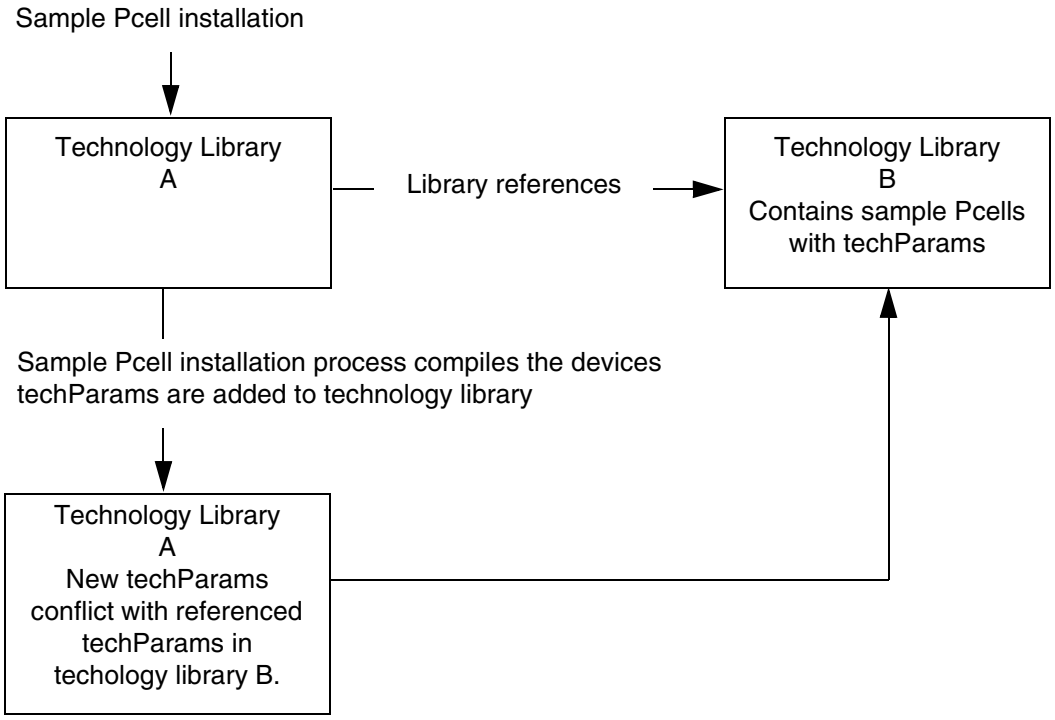
Installing the Sample Parameterized Cells

Before you can use the sample parameterized cells, you must install them into a library. The installation script defines new device classes in your technology file and defines devices for each new class. During the installation process, you have the opportunity to specify the layer definitions and design rules for your technology.

Install Sample Pcells in One Technology Library

Installing sample parameterized cells in different technology libraries across the technology graph can lead to conflicts. When sample parameterized cells are installed, the technology database is updated with the parameters and values (techParams). These parameters and

values must be unique within the technology graph. The following is an example of how conflicts can be created when installing parameterized cells in different technology libraries.



For more information about the technology file in general, refer to the *Technology File and Display Resource File User Guide.* For more information about the changes the installation script makes to your technology file, see "Modifications Made to the Technology File" on page 20.

At any time during the installation process you can

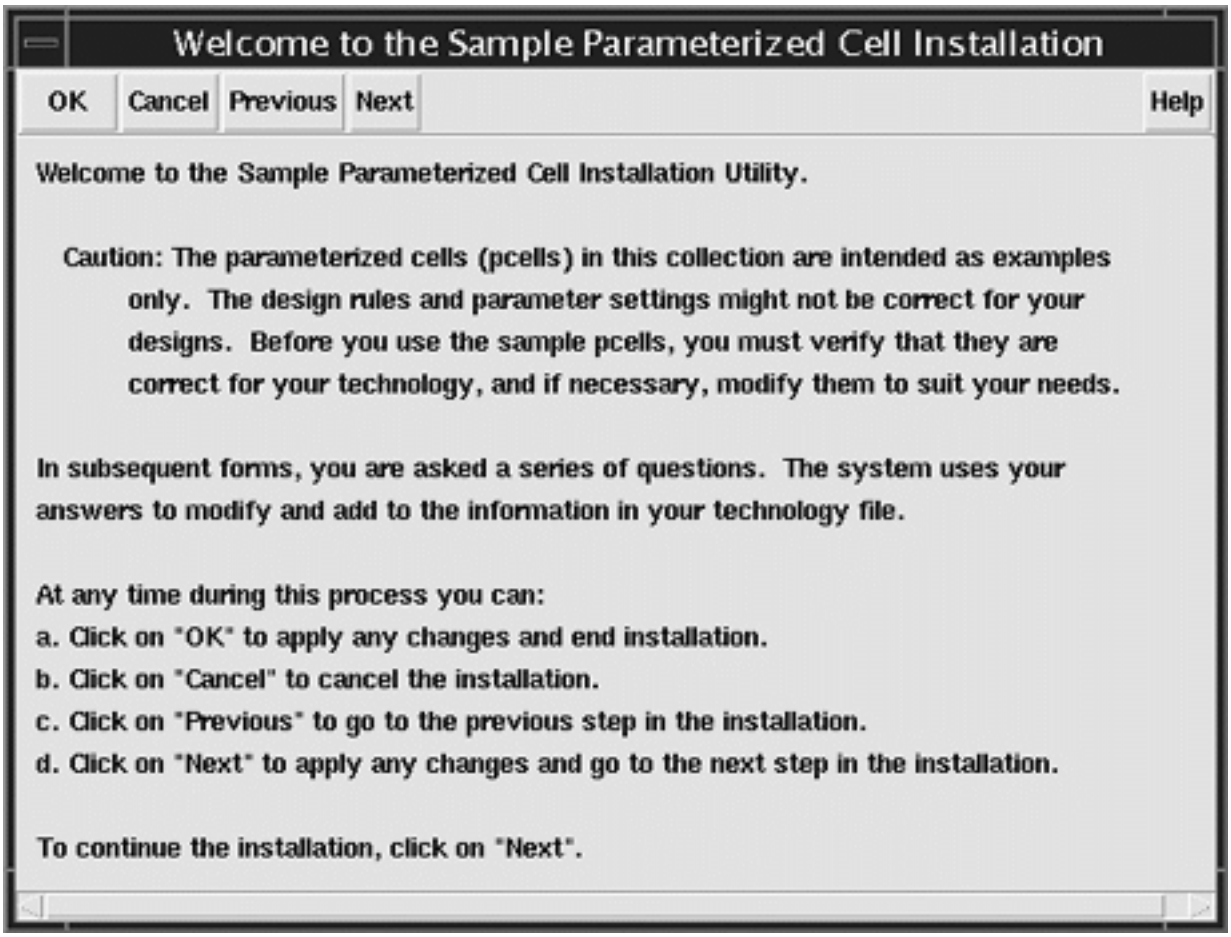
- Click *OK* to apply any changes you have made and exit the installation
- Click *Cancel* to stop the installation process
- Click *Previous* to go to the previous step
- Click *Next* to apply any changes and go to the next step in the installation

To install the sample parameterized cells into a library, follow the steps below.

1. Type the following in the Command Interpreter Window (CIW) input line:

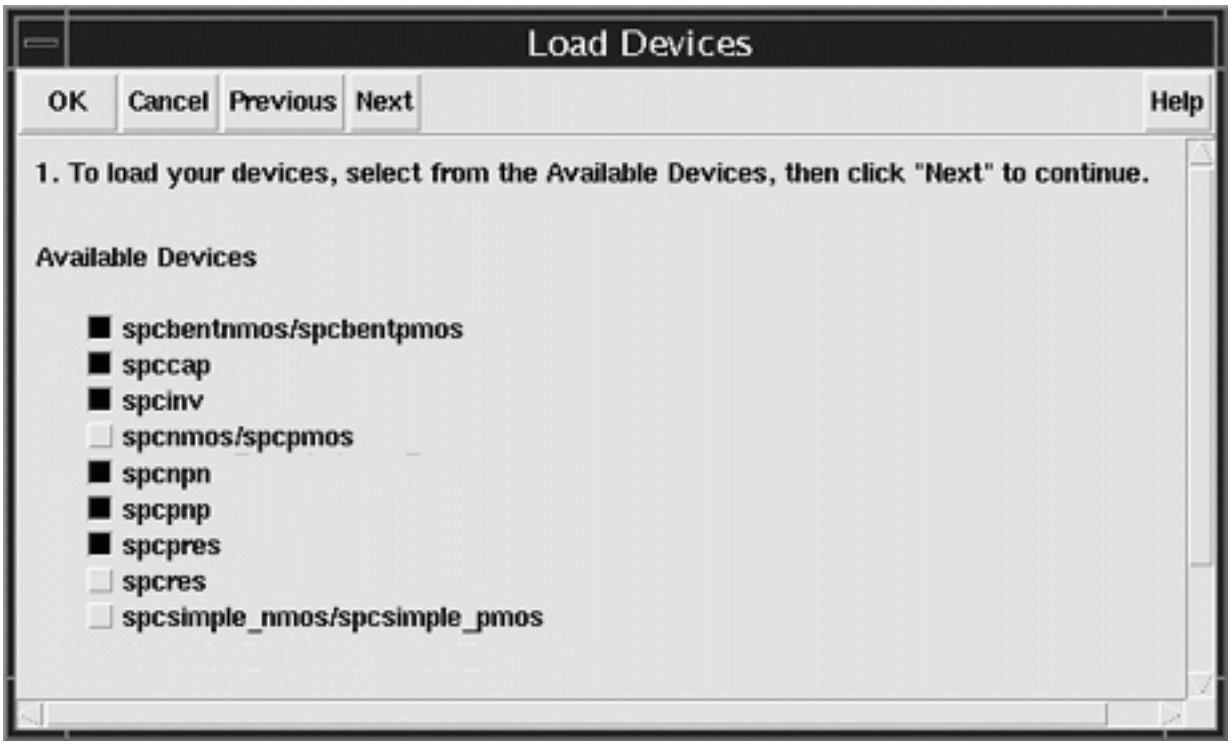
```
setSkillPath(cons(prependInstallPath("samples/ROD/rodPcells")
    getSkillPath()))
load("install/spcLoadInstall.il")
spcInstall()
```

The Welcome to the Sample Parameterized Cell Installation form appears. Read the form before continuing.



2. Click *Next*.

The Load Devices form appears.



Note: Because some of these devices use CMOS technology and some bipolar, you cannot install all of the devices unless there are both CMOS and bipolar layers defined for your technology.

3. Choose the devices you want to install and click *Next*.

   For more information on the devices, see Chapter 2, “Sample Parameterized Cells Reference.”

The Select Destination Library form appears.

4. Choose a library from the *Available Libraries* cyclic field and click *Next*.

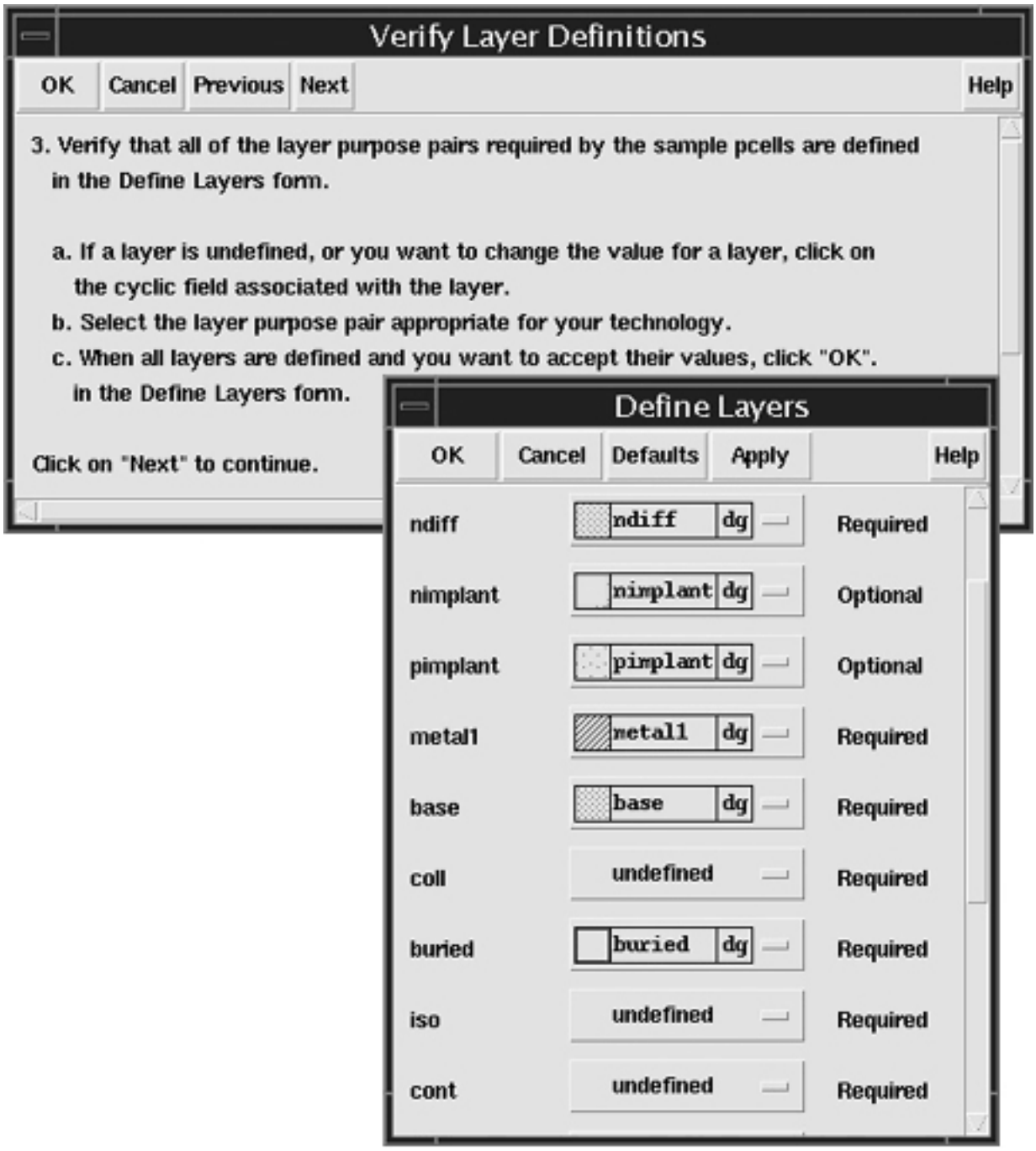
   The SKILL code for the devices is loaded into the directory

   lib/examplePcellLib

   where *lib* is the path to the destination library you selected.

   Only libraries that were created by compiling a technology file will be available for installing the sample Pcells.

The Verify Layer Definitions form and the Define Layers form appear.



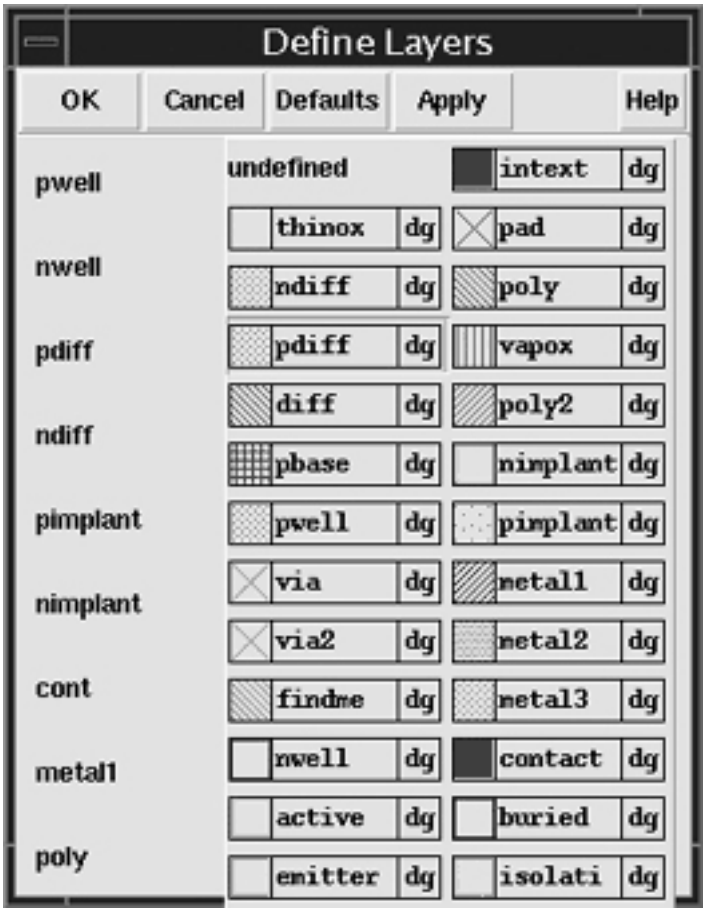
There are three columns in the Define Layers form. The first lists the name of the layers used in the SKILL code for the devices, the second lists the corresponding layer-purpose pairs, and the third lists whether the layers are required or optional. A cyclic field labeled *undefined* identifies layers not previously defined in the technology file of the chosen

library. You must define layer-purpose pairs for all required layers before continuing. You can also use this form to modify previously defined layers of the technology file. The following is a description of the layers used in the Pcell code.

Pcell layer	Layer description
`capVia`	Connects the capBot and capTop layers
`capBot`	Bottom layer of the capacitor
`capTop`	Top layer of the capacitor
`nwell`	n-type well layer
`pwell`	p-type well layer.
`emit`	Emitter layer of the npn device
`iso`	Isolation layer used on the npn, pnp and res devices
`buried`	Buried layer of the npn and pnp devices
`coll`	Collector layer of the npn device and the base layer of the pnp device
`base`	Base layer of the npn device and the collector and emitter layers of the pnp device
`metal1`	Metal layer
`cont`	Contact layer
`pimplant`	p-type implant layer
`nimplant`	n-type implant layer
`ndiff`	n-type diffusion layer
`pdiff`	p-type diffusion layer
`poly`	Polysilicon layer

5. Click the cyclic field to the right of the layer you want to define or modify.

A pop-up menu with several layer choices appears.



a. Choose the layer-purpose pair that corresponds to the layers used in the Pcell code.

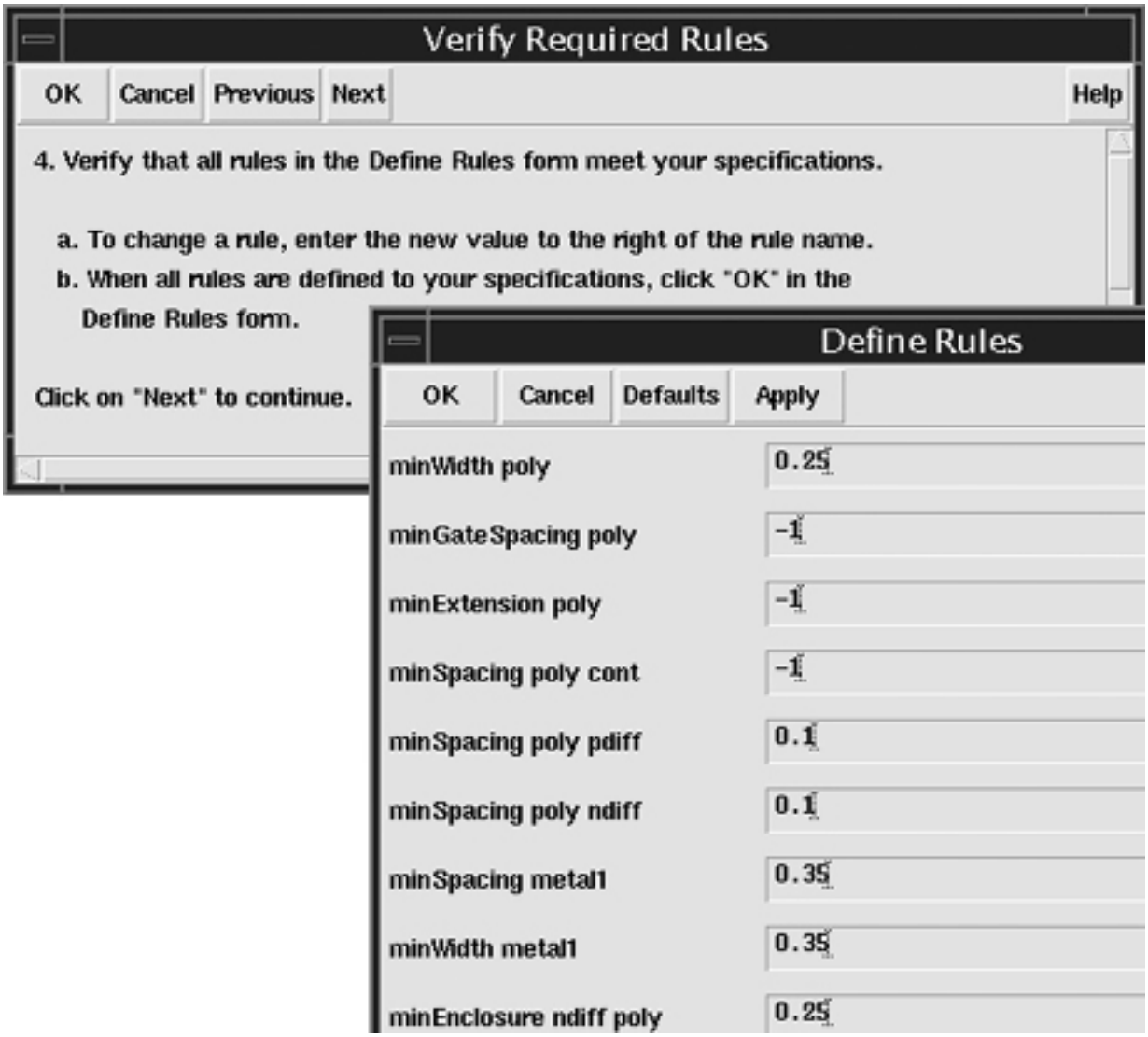
b. Repeat for other layers as necessary.

c. Click *OK* in the Define Layers form.

Note: If you did not select a layer-purpose pair for a required layer, an error message appears. Click *OK* in the dialog box and define all required layers in the Define Layers form.

6. Click *Next* in the Verify Layer Definitions form.

The Verify Required Rules form and the Define Rules form appear.



In the Define Rules form, any design rules that were not previously defined in the technology file default to `-1`.

Caution

You must replace any -1 values with the correct design rule value. Values left at the default of -1 will prevent your technology file or your Pcell from evaluating.

The rules in the Define Rules form and the values you assign to them are added to the `techParams` subclass of the `controls` class of the technology file. For more

information on design rules and the technology file, refer to the *Technology File and Display Resource File User Guide.*

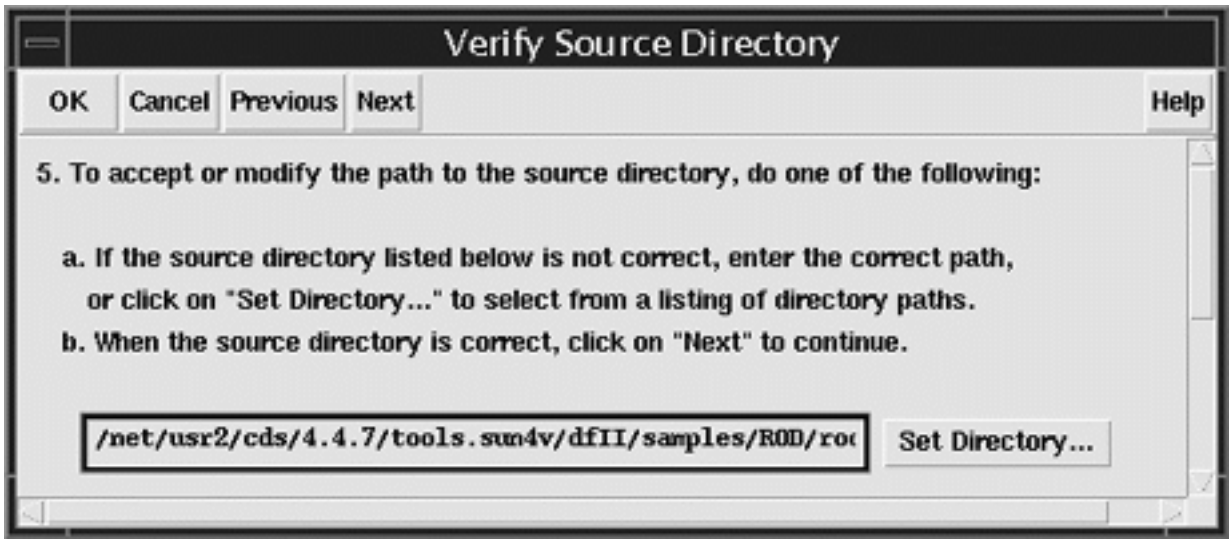
7. Click in the field to the right of the rule you want to modify.
   a. Type the correct design rule for your technology in the field.
   b. Repeat for all design rules requiring modification.
   c. Verify that there are no design rules with values of -1.
   d. When you are finished, click *OK* in the Define Rules form.

Caution

Modifying the rules changes the design rules saved to the technology file and affects all applications that use those rules.

8. Click *Next* in the Verify Required Rules form.

   The Verify Source Directory form appears.



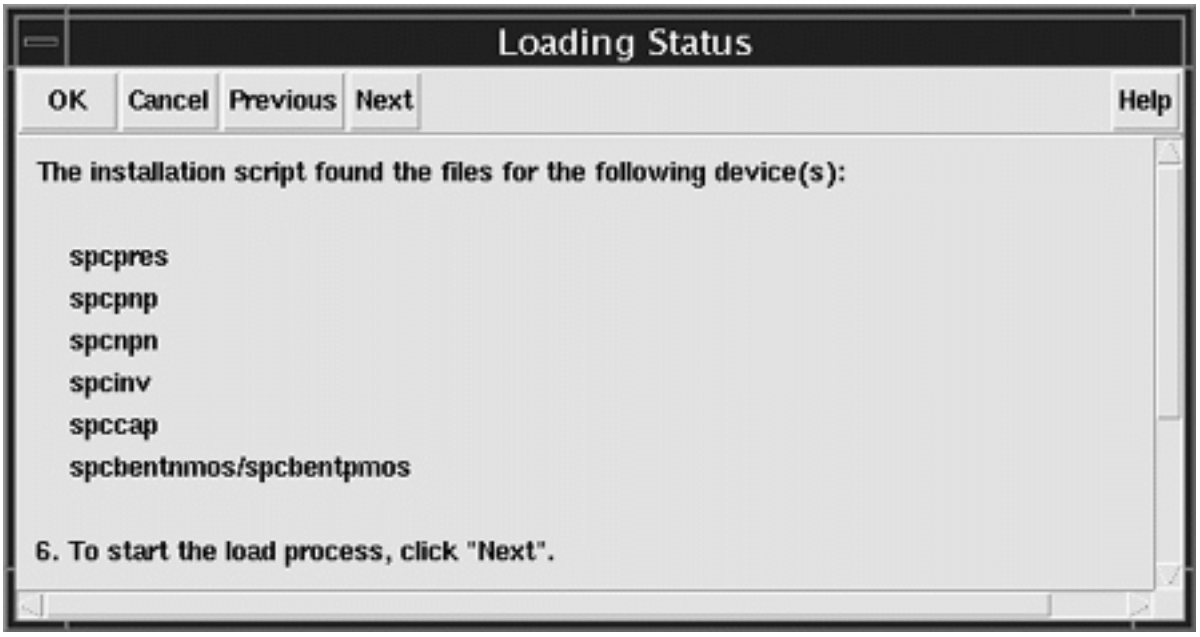
   In this form, you specify which directory contains the source files for the devices you want to install. By default, this is

   your_install_dir`/tools/dfII/samples/ROD/rodPcells`

9. (Optional) If you have copied and modified the default source files, type the path to the modified files in the field, or click *Set Directory* to browse for the correct path.
10. Click *Next*.

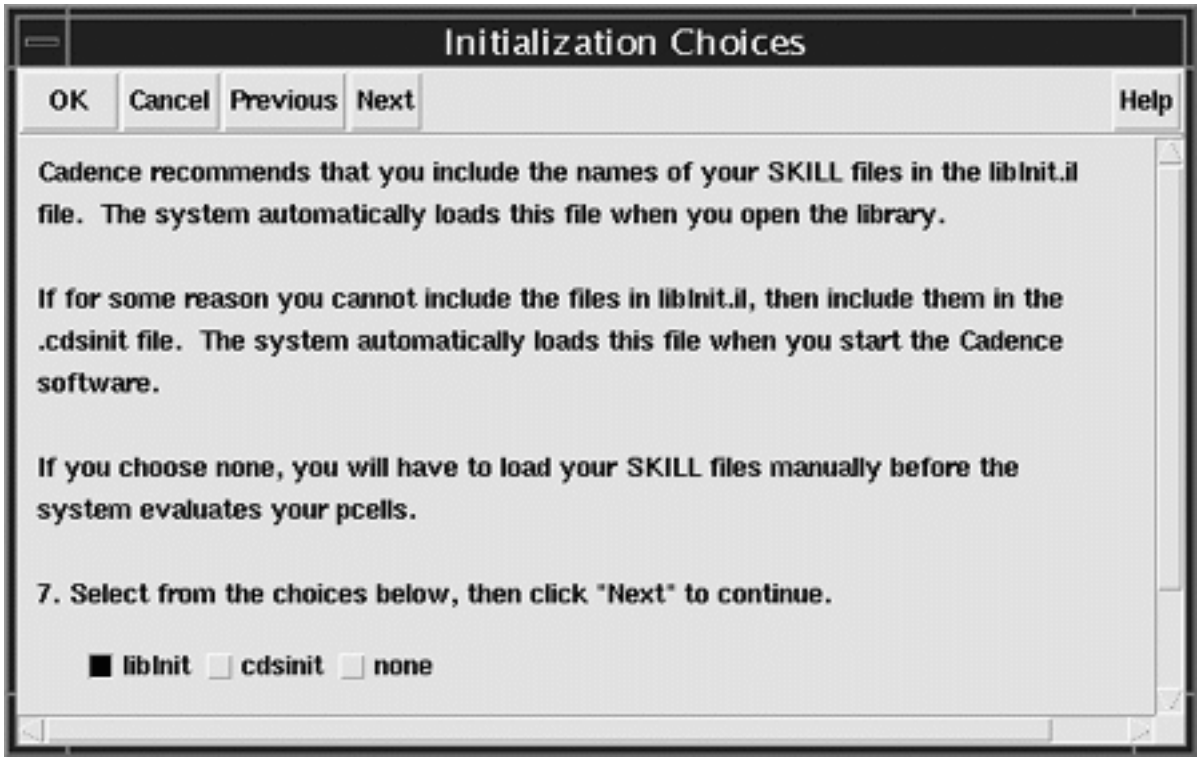
Note: If you specified an invalid path, the Install Path Error dialog box appears. Click *OK* in the dialog box and specify a valid path.

The Loading Status form appears and confirms that the files for the devices you chose to install were found in the directory.



11. Click *Next*.

The Initialization Choices form appears.



In this form, you choose where to include instructions for loading the Cadence SKILL language files defining the devices.

- *libInit*

  Choosing `libInit.il` creates a file in the library directory or adds additional information to an existing `libInit.il` file. All applications read the `libInit.il` file, which enables all applications and environments to evaluate the sample Pcells. For more information about the `libInit.il` file, refer to "Loading Pcells from the libInit.il File" on page 19.

- *cdsinit*

  Choosing `.cdsinit` creates a file in the library directory, or adds additional information to an existing `.cdsinit` file. When loading Pcells from the `.cdsinit` file, the sample Pcells are loaded when you start the software.

  Not all applications read the `.cdsinit` file. Sample Pcells will not be evaluated for translation in background mode by applications that do not use the `.cdsinit` file. For any background translation that does not use the `.cdsinit` file, you need to explicitly load the SKILL code for the devices.

- *none*

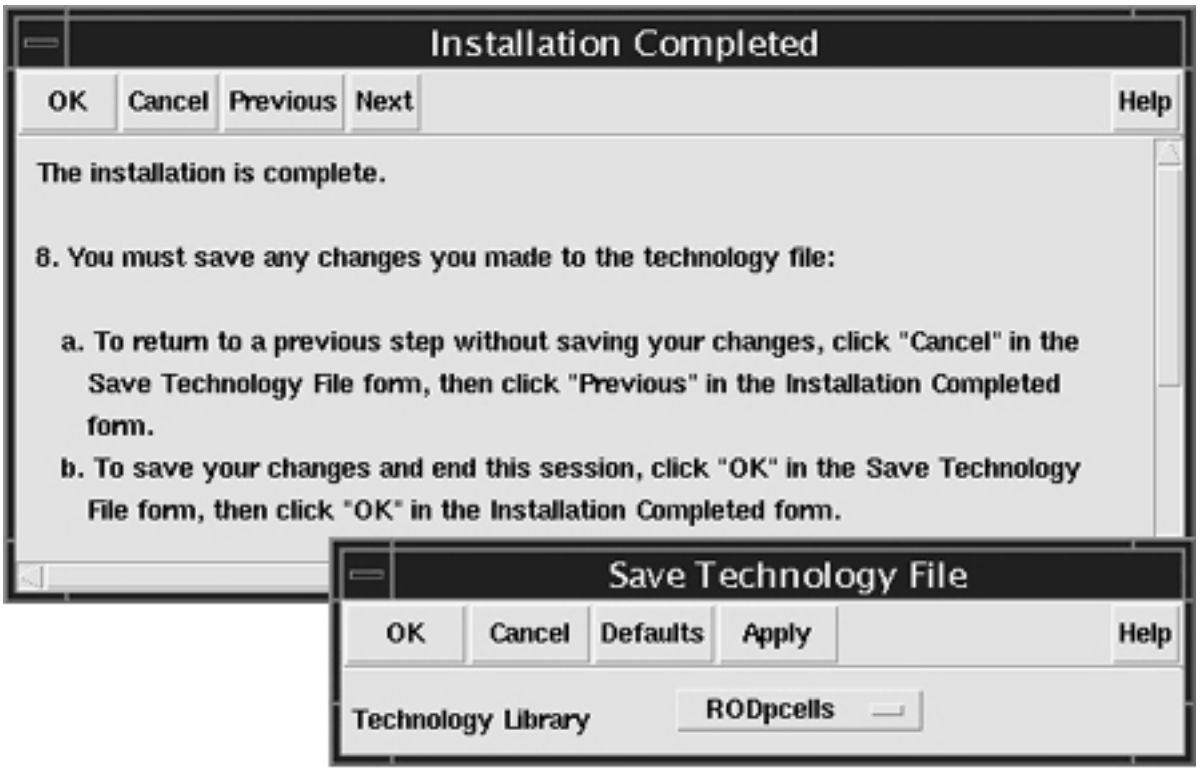
  Choosing none requires you to load the SKILL files manually before the system can evaluate your Pcells. The following example loads the *bentmos* sample Pcell.

```
load(strcat(ddGetObjReadPath(ddGetObj("RODpcells"))
"/examplePcellLib/spcLoadUtilities.il"))
load(strcat(ddGetObjReadPath(ddGetObj("RODpcells"))
"/examplePcellLib/utility.il"))
load(strcat(ddGetObjReadPath(ddGetObj("RODpcells"))
"/examplePcellLib/bentmos.callback"))
load(strcat(ddGetObjReadPath(ddGetObj("RODpcells"))
"/examplePcellLib/bentmos.il"))
```

Note: To load Pcells for background translation using XStream, use the *User SKILL File*. For more information, refer to "Design Translation Using XStream translator" in the *Design Data Translator's Reference on OpenAccess*.

12. Make a selection and click *Next*.

The Installation Completed form and the Save Technology File form appear.



The changes you make to the layer definitions and design rules during this installation session override any previous changes you made to the technology file.

13. In the Save Technology File form, click *OK* to save your changes to the library shown in the cyclic field.

    A confirmation dialog appears.

14. In the confirmation dialog, click *Yes*.

    The installation is complete.

15. Click *OK* to exit the installation utility.

Validating the Installation

To validate the installation of the sample cells, follow these steps:

1. Open the Library Browser and verify that the new device masters exist in the destination library.
2. Place each of the devices in a new layout view and verify that they are created properly.
3. If you were prompted to save your technology file, exit and then restart the Cadence software to verify that the modified technology file and the `libInit.il` or `.cdsinit` files load correctly.

Loading Pcells from the libInit.il File

When loading Pcells from the `libInit.il` file, the system loads the sample Pcells when the library is accessed (the first time a cellview is opened). If the sample Pcells are loaded into multiple libraries and from your `libInit.il`, the first time each of the libraries in which the sample Pcells have been installed is accessed, the Pcells will be evaluated. This causes multiple messages about redefined functions to appear in the CIW. These redefined messages can safely be ignored. To avoid multiple redefined messages, you can add

additional SKILL code to your `libInit.il` files. The following example adds an `unless(boundp( 'SPCLOADED)` statement before the first line of code that was added by the installation program and sets the global variable `SPCLOADED to t` after the last device load call.

```
unless( boundp( 'SPCLOADED) ; Adds global variable and checks if bound.
load(strcat(ddGetObjReadPath(ddGetObj("designLib"))
"/examplePcellLib/spcLoadUtilities.il"))
load(strcat(ddGetObjReadPath(ddGetObj("designLib"))
"/examplePcellLib/utility.il"))
load(strcat(ddGetObjReadPath(ddGetObj("designLib"))
"/examplePcellLib/mos.il"))
load(strcat(ddGetObjReadPath(ddGetObj("designLib"))
"/examplePcellLib/mos.callback"))
SPCLOADED = t
); Sets the global variable to true if Pcells are loaded.
```

The use of the `libInit.il` file has some restrictions. SKILL code loaded by the `libInit.il` file should only assume that 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 the SKILL Language Reference Manual; for example: `car`, `if`, `foreach`, `sprintf`, `while`.
- SKILL functions from the following families: `db dd cdf rod tech`

No SKILL code loaded by the `libInit.il` file should be dependent on graphical SKILL functions like `hiSetBindKey`.

To call functions or load files that use graphical SKILL functions, for example, from the `hi`, `ge`, `le`, `sch` families, make the code conditional with `isCallable`.

For example:

```
if(isCallable('hiSetBindKey)
     hiSetBindKey(....)
     ...
     )
```

Modifications Made to the Technology File

The installation script makes several modifications to your technology file.

Layer Correspondence Parameters in techParams

The installation script creates a table in the `techParams` subclass that lists the generic layer functions used in the parameterized cells and the corresponding layer numbers in your particular technology. You define this data when you fill in the Define Layers form.

Note: The `techParams` subclass may also contain router translation rules that are unrelated to these sample parameterized cells.

Below is an example of this table of parameters and values.

```
;( parameter          value              )
;( ----------         ------------------ )
  ( poly              ("poly1" "drawing") )
  ( metal1            ("metal1" "drawing") )
  ( cont              ("contact" "drawing") )
  ( nimplant          ("txtndiff" "drawing") )
  ( pimplant          ("txtpdiff" "drawing") )
  ( ndiff             ("ndiff" "drawing") )
  ( pdiff             ("pdiff" "drawing") )
  ( nwell             ("nwell" "drawing") )
  ( pwell             ("viaE" "drawing") )
```

The `techParams` subclass also receives several variables that define characteristics of the sample devices:

This Variable...	Is Used in This Cell...
`minMosWidth`	`spcnmos`, `spcpmos`
`minRes`	`spcres`
`emitArea`, `emitBaseSpacing`	`spcnpn`
`minCap`, `capCoeff`, `unitC`, `maxCap`	`spccap`

Physical DRC Rules

The `minEnclosure`, `minSpacing`, and `minWidth` rules in the `physicalRules` section of the technology file are used to set default values in the Define Rules form. The installation script adds additional rules to the `physicalRules` section for any design rules that were not previously defined in the technology file. These rules are shared by other Cadence design framework II applications.

If you modify the rules while installing the cells, you affect all applications that use the rules.

Device Classes

In the `devices` section of the technology file, the installation script creates device classes for each cell. The class for the *spcbentnmos* and *spcbentpmos* devices is `syBGEnhancement`. The classes for the other devices begin with the prefix `spc` and follow the naming of the devices themselves. For example:

```
tcCreateDeviceClass( "layout" "spcsimple_mos" )
```

During the installation script, you are prompted to install SKILL functions for the cells in the `libInit.il` file (or in your `.cdsinit` file). The device classes in the technology file call these SKILL functions.

2

Sample Parameterized Cells Reference

This chapter discusses the following:

Modifying Pcell Parameter Values

Once you have installed the sample Pcells, you can place instances of them in your layout and modify their parameters to suit your specifications. When you place an instance using the Create Instance form (choose *Create – Instance*), the system prompts you to enter values for a variety of parameters or to accept the defaults. You can modify an instance using the Edit Properties form (choose *Edit – Properties*) as well.

Each section in this chapter consists of a brief discussion of a sample Pcell and a description of its parameters as they appear in the Create Instance and Edit Properties forms. Each parameter entry contains information about the purpose and default value of the parameter. When appropriate, some entries also provide information about valid parameter values and examples of such values.

Default Values

During the installation, you assigned values to design rules and then saved them to the `techParams` subclass of the `controls` class of the technology file. The default values for many of the parameters of the sample Pcells are derived from these same values.

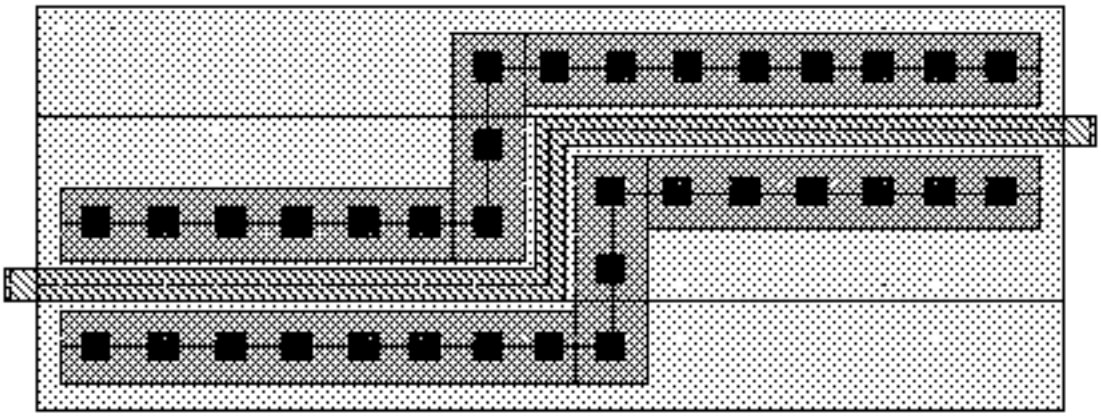
For instance, the gate length of the spcbentnmos device defaults to the minimum width of the poly layer (`minWidth poly`), which you specified during the installation. The values that you can specify for many of the parameters are bounded by these same technology file values.

spcbentnmos, spcbentpmos

The spcbentnmos device is an NMOS transistor with a bent-gate option, and the spcbentpmos device is the PMOS equivalent. The parameter *Coordinate list* allows you to modify the path of the gate by specifying the coordinates of the center of the gate.

This section discusses the spcbentnmos/spcbentpmos device parameters that appear in the Create Instance (choose *Create – Instance*) and Edit Properties (choose *Edit – Properties*) forms of the Virtuoso® layout editor. The default values for many of the parameters come from your technology file.

The spcbentpmos Device, Placed with Source and Drain Contacts, and Coordinate List Set to (0.0:0.0 10.0:0.0 10.0:3.0 20.0:3.0)



Parameters

Gate Width	Controls the width of the gate. When *Coordinate list* has values specified, *Gate Width* is derived from those values. Valid Values: a number in meters, in engineering notation, greater than or equal to the value of `minMosWidth` Example: `4e-6` Default: the value of `minMosWidth`
Gate Length	Controls the length of the gate. Valid Values: a number in meters, in engineering notation, greater than or equal to the value of `minWidth poly` Example: `6e-7` Default: the value of `minWidth poly`
Multiplication Factor	The Virtuoso layout accelerator (Virtuoso XL) supports the use of the multiplication factor (m-factor) as a parameter to define a one-to-many parallel relationship between a device in a schematic and multiple instances of the device in the layout. This parameter is intended for use at the schematic level; if you change the m-factor of a layout view, the system multiplies the width of the device by the specified factor. Valid Values: a positive floating-point number Default: `1`
Coordinate list	Defines the X and Y coordinates of the gate path. Valid Values: a list of X:Y coordinates Example: `(0.0:0.0 10.0:0.0 10.0:3.0 20.0:3.0)` Default: none (the gate is straight)
Source Contacts?	Adds source contacts. The *Source Contact Position* and *Source Contact Coverage* parameters appear when *Source Contacts?* is on. Default: off
Drain Contacts?	Adds drain contacts. The *Drain Contact Position* and *Drain Contact Coverage* parameters appear when *Drain Contacts?* is on. Default: off

Source Contact Position

Positions the contacts vertically within the source. When the contacts are not spread out evenly along the width of the source, you can shift the contacts toward the top, center, or bottom of the source, relative to the unrotated device.
Valid Values: *Center*, *Top*, or *Bottom*
Default: *Center*

Drain Contact Position

Positions the contacts vertically within the drain. When the contacts are not spread out evenly along the width of the drain, you can shift the contacts toward the top, center, or bottom of the drain, relative to the unrotated device.
Valid Values: *Center*, *Top*, or *Bottom*
Default: *Center*

Source Contact Coverage

Controls the percentage of the width of the source that the contacts cover. This lets you route other metal lines across the device without connecting them to it electrically.
Valid Values: `10` to `100`
Default: `100`

Drain Contact Coverage

Controls the percentage of the width of the drain that the contacts cover. This lets you route other metal lines across the device without connecting them to it electrically.
Valid Values: `10` to `100`
Default: `100`

Bulk Node Connection(s)

Connects the bulk-substrate to source or drain.
Valid Values: *S* for source, *D* for drain, or blank for neither
Default: *S*

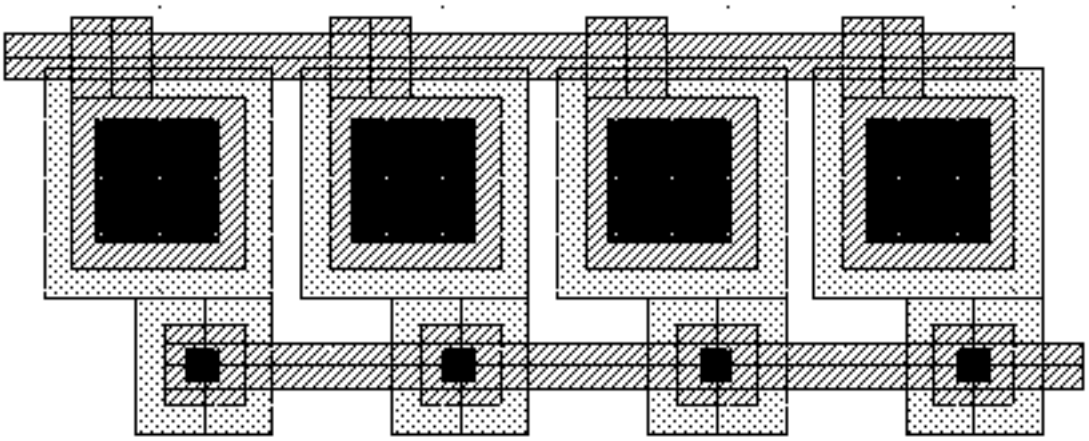
spccap

The spccap device is an array capacitor. You can change the aspect ratio of the array by adjusting the number of rows and columns. You can round up the unit capacitance value or add an extra capacitor to the array to achieve the precise required capacitance value.

This section discusses the spccap device parameters that appear in the Create Instance (choose Create – Instance) and Edit Properties (choose *Edit – Properties*) forms of the

Virtuoso layout editor. The default values for many of the parameters come from your technology file.

The spccap Device, Placed in a Four-Column Array Layout



Parameters

Capacitance	Controls the required capacitance. Valid Values: a number in farads, in engineering notation, greater than or equal to the value of `minCap` and less than or equal to the value of `maxCap` Example: `1e-4` Default: the value of `minCap`
Actual Capacitance	Specifies the total capacitance of all the unit capacitors in the Pcell. This is a noneditable field.
Array Type Layout?	Builds the Pcell of multiple-unit capacitor tiles. Default: on
Connect Capacitor Tiles	Connects the unit capacitors within the Pcell. Default: off
Define Array By	Controls the internal layout of multiple-unit capacitors within the Pcell. Valid Values: *Square*, *Rows*, or *Columns* Default: *Square*
Number of Rows	Controls the number of rows of unit capacitors within the Pcell. Valid Values: a positive integer Default: `1`
Number of Columns	Controls the number of columns of unit capacitors within the Pcell. Valid Values: a positive integer Default: `1`
Unit Size Capacitance	Controls the required capacitance for the unit capacitors within the Pcell. Valid Values: a number in farads, in engineering notation Default: the value of `unitC`
Cap Round Up?	Rounds the capacitance value to the next higher unit of capacitance. *Cap Round Up?* and *Add Extra* cannot both be on at the same time. Default: on

Add Extra

Adds an extra capacitor to achieve the required capacitance. *Add Extra* and *Cap Round Up?* cannot both be on at the same time.
Default: off

Multiplication Factor(m)

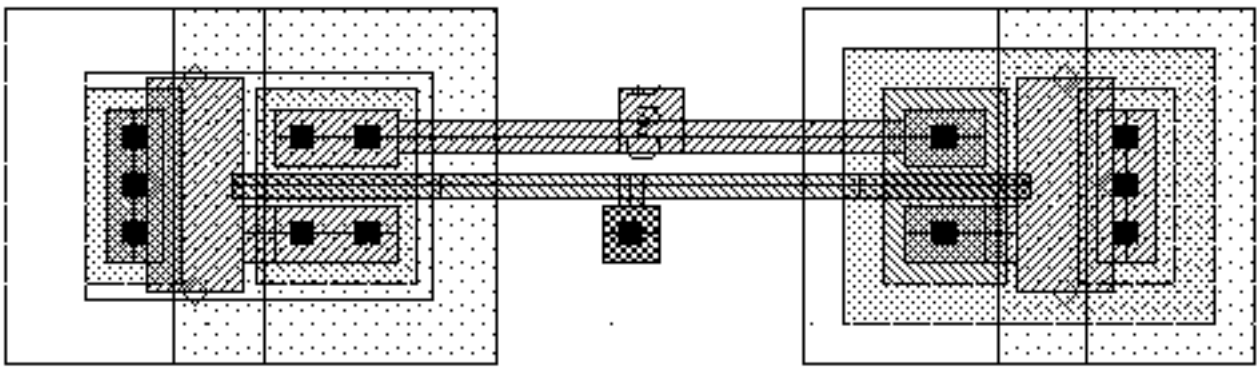
Virtuoso XL supports the use of the multiplication factor (m-factor) as a parameter to define a one-to-many parallel relationship between a device in a schematic and multiple instances of the device in the layout. This parameter is intended for use at the schematic level.

spcinv

The spcinv device is an inverter built using the spcnmos and the spcpmos devices. You can modify the gate width and length of both transistors, define the supply width, and set the height of the whole cell. Note the stretch handles that let you graphically scale the device.

This section discusses the spcinv device parameters that appear in the Create Instance (choose *Create – Instance*) and Edit Properties (choose *Edit – Properties*) forms of the Virtuoso layout editor. The default values for many of the parameters come from your technology file.

The spcinv Device, Placed with 90° of Rotation



Parameters

nMos Gate Width	Controls the width of the gate of the NMOS transistor in the inverter. Valid Values: a number in meters, in engineering notation Example: `3e-6` Default: the value of `minMosWidth ndiff`
nMos Gate Length	Controls the length of the gate of the NMOS transistor in the inverter. Valid Values: a number in meters, in engineering notation Example: `1e-7` Default: the value of `minWidth poly`
nMos fingers	Controls the number of fingers in the NMOS transistor. Valid Values: a positive integer Default: `1`
pMos Gate Width	Controls the width of the gate of the PMOS transistor in the inverter. Valid Values: a number in meters, in engineering notation Example: `4e-6` Default: the value of `minMosWidth pdiff`
pMos Gate Length	Controls the length of the gate of the PMOS transistor in the inverter. Valid Values: a number in meters, in engineering notation Example: `1e-7` Default: the value of `minWidth poly`
pMos fingers	Controls the number of fingers in the PMOS transistor. Valid Values: a positive integer Default: `1`
Supply Width	Controls the width of the power and ground lines. Valid Values: a number in meters, in engineering notation Example: `2.4e-6` Default: the value of `minWidth metal1 * 3`
Cell Height	Controls the distance from the bottom of the ground supply to the top of the power supply. Valid Values: a number in meters, in engineering notation Example: `2.2e-5` Default: the value of `minWidth metal1 * 30`

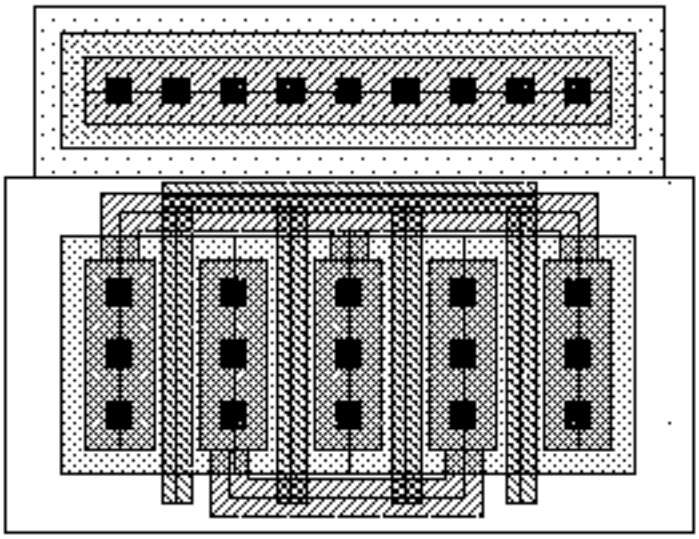
Add substrate contacts?	Adds substrate contacts. Default: on
Cell Horiz. Pitch	Controls the length of device. The system modifies the length of the supply lines to be a multiple of the cell horizontal pitch value. Valid Values: a number in meters, in engineering notation Example: `5e-8` Default: the value of `mfgGridResolution`

spcnmos, spcpmos

The spcnmos device is an n-diffusion straight-gate MOS transistor, and the spcpmos device is the p-diffusion equivalent. You can modify the gate width and length of the transistor, apply fingered or threshold gate splitting, add contacts to either side of the gate, and specify the position of the source and drain contacts.

This section discusses the spcnmos/spcpmos device parameters that appear in the Create Instance (choose *Create – Instance*) and Edit Properties (choose *Edit – Properties*) forms of the Virtuoso layout editor. The default values for many of the parameters come from your technology file.

The spcnmos Device, Placed with Left, Right, and Substrate Contacts and a Four-Fingered Split Gate



Parameters

Gate Width	Controls the width of the gate. Valid Values: a number in meters, in engineering notation, greater than or equal to the value of `minMosWidth` Example: `4e-6` Default: the value of `minMosWidth`
Gate Length	Controls the length of the gate. Valid Values: a number in meters, in engineering notation, greater than or equal to the value of `minWidth poly` Example: `1e-6` Default: the value of `minWidth poly`
Multiplication Factor(m)	Virtuoso XL supports the use of the multiplication factor (m-factor) as a parameter to define a one-to-many parallel relationship between a device in a schematic and multiple instances of the device in the layout. This parameter is intended for use at the schematic level; if you change the m-factor of a layout view, the system multiplies the width of the device by the specified factor. Valid Values: a positive floating-point number Default: `1`
Gate Splitting	Splits the gate. The *Type of Splitting*, *Number of Fingers* (or *Threshold Value* if *Type of Splitting* is set to *Threshold*), and *Internally connect poly?* parameters appear when *Gate Splitting* is on. Default: off
Type of Splitting	Controls how the gate is split. Valid Values: *Fingered* or *Threshold* Default: *Fingered*
Number of Fingers	Controls the number of fingers in the transistor. Valid Values: a positive integer Default: `1`
Threshold Value	Specifies the maximum width for each gate when *Gate Splitting* is on. Valid Values: a number in meters, in engineering notation, greater than or equal to the value of `minMosWidth * 10` Example: `30` Default: the value of `minMosWidth * 10`

Left Contacts	Adds contacts to the left of each gate relative to the unrotated device. The left contact is always the source contact. If you set *Number of Fingers* to an even number, the source contacts and drain contacts alternate, starting with a source contact to the left of the left-most gate (relative to the unrotated device). Default: on
Right Contacts	Adds contacts to the right of each gate relative to the unrotated device. If you set *Number of Fingers* to an odd number, the right contact is a drain. If you set *Number of Fingers* to an even number, the source contacts and drain contacts alternate, starting with a source contact to the left of the left-most gate (relative to the unrotated device). Default: on
Add substrate contact?	Adds substrate contacts. The *Substrate Contact Position* parameter appears when *Add substrate contact?* is on. If *Bulk Node Connection* is set to *S*, the substrate contact is positioned to the left of the gate, relative to the unrotated device. If *Bulk Node Connection* is set to *D*, the substrate contact is positioned to the right of the gate, relative to the unrotated device. Default: off
Substrate Contact Position	Positions the substrate contact relative to the unrotated device. Valid Values: *Top* or *Bottom* Default: *Top*
Add substrate well?	Adds a substrate well. Default: off
Bulk Node Connection	Connects the bulk substrate to the source or drain. Valid Values: *S* for source, *D* for drain, or blank for neither Default: *S*
Internally connect poly?	Connects the poly gates to each other within the Pcell. Default: off
Connect poly	Positions the poly gate connection relative to the unrotated device. Valid Values: *Top* or *Bottom* Default: *Top*

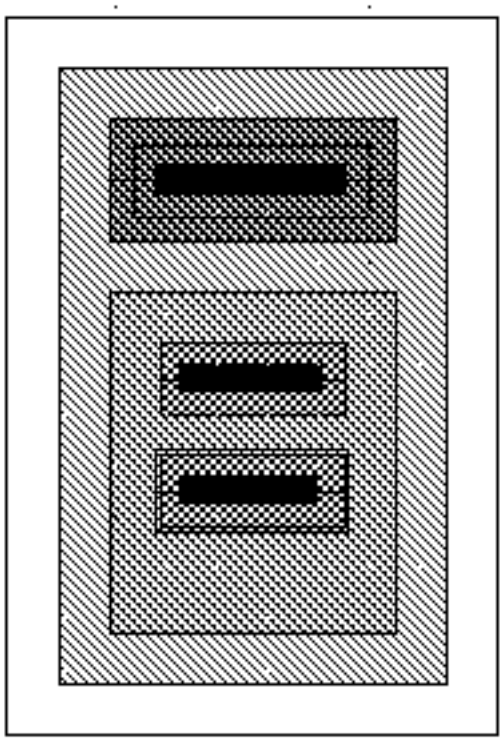
Connect S/D to metal? Connects the source or the drain to metal.
Valid Values: *None*, *Source*, *Drain*, or *Source&Drain*
Default: *None*

spcnpn

The spcnpn device is an NPN bipolar transistor that, given the wide variety of bipolar transistor designs, has been kept fairly simple. There are five choices of collector, emitter, and base configurations and a parameter for the emitter width. Note the stretch handles that let you graphically scale the device.

This section discusses the spcnpn device parameters that appear in the Create Instance (choose *Create – Instance*) and Edit Properties (choose *Edit – Properties*) forms of the Virtuoso layout editor. The default values for many of the parameters come from your technology file.

The spcnpn Device, Placed in the CBE Configuration



Parameters

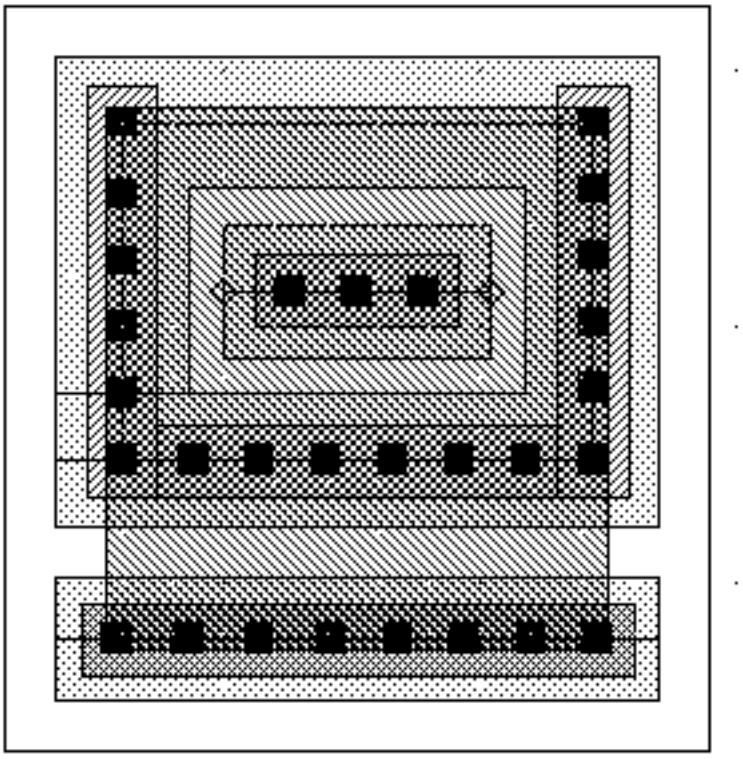
NPN Type	Chooses the collector-base-emitter configuration. `Valid Values:` *CBE*, *CEB*, *CEBC*, *CBEC*, `or` *CEBEC* `Default:` *CBE*
Emitter Width	Controls the width of the emitter. Valid Values: a number in meters, in engineering notation Example: `2.6e-6` Default: the value of `2 * (2 * minEnclosure emit cont + minWidth cont)`
Multiplication Factor	Virtuoso XL supports the use of the multiplication factor (m-factor) as a parameter to define a one-to-many parallel relationship between a device in a schematic and multiple instances of the device in the layout. This parameter is intended for use at the schematic level.

spcpnp

The spcpnp device is a stretchable PNP bipolar transistor that, given the wide variety of bipolar transistor designs, has been kept fairly simple. The Pcell contains the emitter and the base. The collector is a zero-level multipart path built with the ROD function `rodCreatePath`.

This section discusses the spcpnp device parameters that appear in the Create Instance (choose *Create – Instance*) and Edit Properties (choose *Edit – Properties*) forms of the Virtuoso layout editor. The default values for many of the parameters come from your technology file.

The spcpnp Device



Parameters

P+ Emitter Width	Controls the width of the gate. Valid Values: a number in meters, in engineering notation Example: `6e-6` Default: the value of `2 * (2 * minEnclosure base cont + minWidth cont)`
Multiple emitters	Controls the number of emitters in the Pcell. Valid Values: a positive integer Default: `1`
Emitter access direction	Selects the access direction of the emitter. Valid Values: *Top*, *Left*, *Right*, *Bottom*, or *None*. Default: *Top*
Multiplication Factor	Virtuoso XL supports the use of the multiplication factor (m-factor) as a parameter to define a one-to-many parallel relationship between a device in a schematic and multiple instances of the device in the layout. This parameter is intended for use at the schematic level.

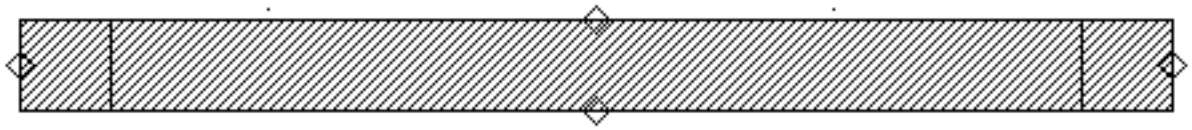
spcpres

Sometimes resistors in a schematic represent estimates of the resistance that certain signals might have after the layout designer draws the design. Including these resistors helps improve the simulation results. During layout, the designer does not draw the resistor because it represents the resistance of the shapes needed to connect the other components.

Virtuoso XL expects the layout to match the schematic device for device and net for net. The spcpres device was designed to let you place a resistor in the schematic and connect separate nets to each side of the device, without causing Virtuoso XL to expect to see a corresponding resistor in the layout.

The spcpres device uses the sheet resistance (`sheetRes`) from the technology file for the metal layer and adjusts the device size accordingly. Given the low resistance of the metal layer, you need to enter relatively large values to see a visible change in the layout view. The pins are never placed more than twice the minimum grid spacing apart. The separation between the pins prevents the Virtuoso XL connectivity extractor from reporting a short where the two nets touch. Note the stretch handles that let you graphically scale the device.

The spcpres Device



Parameters

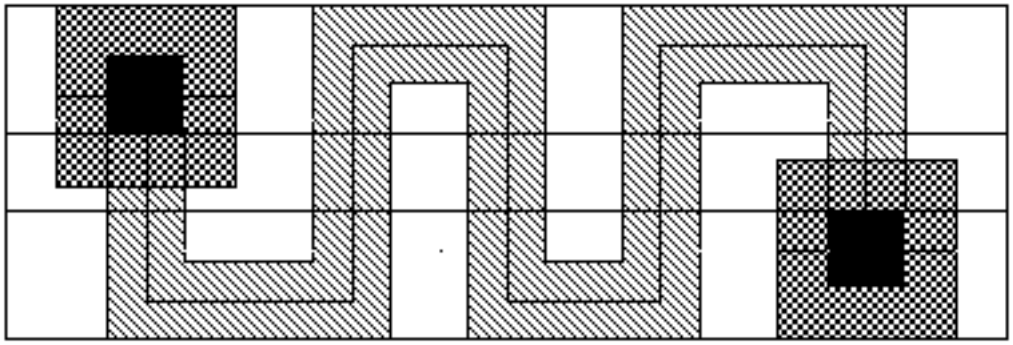
Resistor type	Selects the type of resistor material. Valid Values: *poly*, *ndiff*, *pdiff*, or *metal1* Default: *metal1*
Input value	Specifies the input that defines the resistor. Valid Values: *Length* or *Resistance* Default: *Resistance*
Resistance	Controls the total resistance of the device. Valid Values: a number in ohms, in engineering notation Example: `8e+3` Default: the value of `8 * minRes`
Length	Controls the length of the resistor path. Valid Values: a number in meters, in engineering notation Default: varies based on the values of `minRes` and the values of `sheetRes` and `minWidth` of the specified resistor type.
Width	Controls the width of the resistor path. Valid Values: a number in meters, in engineering notation Default: the value of `minWidth` for the resistive layer
Multiplication Factor	Virtuoso XL supports the use of the multiplication factor (m-factor) as a parameter to define a one-to-many parallel relationship between a device in a schematic and multiple instances of the device in the layout. This parameter is intended for use at the schematic level. Default: `1`

spcres

The spcres device is a resistor built to let you define the type of resistor material (poly, n-diffusion, or p-diffusion), resistance, length, and width. You can draw the resistor in a serpentine or strap formation for layout optimization.

This section discusses the spcres device parameters that appear in the Create Instance (choose *Create – Instance*) and Edit Properties (choose *Edit – Properties*) forms of the Virtuoso layout editor. The default values for many of the parameters come from your technology file.

The spcres Device, Placed in a Five-Segment Serpentine Shape



Parameters

Resistor type	Selects the type of resistor material. Valid Values: *poly*, *ndiff*, or *pdiff* Default: *poly*
Resistance	Controls the total resistance of the device. Valid Values: a number in ohms, in engineering notation Example: `8e+3` Default: the value of `8 * minRes`
Length	Controls the length of the resistor path. Valid Values: a number in meters, in engineering notation Default: varies based on the values of `minRes` and the values of `sheetRes` and `minWidth` of the specified resistor type.
Input value	Specifies the input that defines the resistor. Valid Values: *Length* or *Resistance* Default: *Resistance*
Width	Controls the width of the resistor path. Valid Values: a number in meters, in engineering notation Default: the value of `minWidth poly`
Draw resistor as	Selects the shape of the resistor. Choose *Straps* to draw the resistor with the number of resistor paths specified in *Number of segments* between the contacts. Choose *Serpentine* to draw the resistor path in an accordion shape with the specified number of segments. Examples: If you choose *Straps* and set *Number of segments* to `5`, the total resistor value is split evenly among five individual resistors, which are connected in series. If you choose *Serpentine* and set *Number of segments* to `5`, the system draws a single, five-segment, accordion-shaped resistor. Valid Values: *Straps* or *Serpentine* Default: *Straps*

Number of segments	Controls the number of straps or serpentine segments. When you set *Draw resistor as* to *Straps*, *Number of segments* controls how many resistor paths the system draws between the serially connected contacts. When you set *Draw resistor as* to *Serpentine*, *Number of segments* controls how many segments are in the accordion shape of the resistor path. Valid Values: an integer, with a minimum value of `1` for straps and `2` for serpentine Default: `1`
Serpentine width	Controls the width of the serpentine resistor path. Valid Values: a number in meters, in engineering notation Default: varies based on the values of `minWidth cont`, `minEnclosure metal1 cont`, and the `minSpacing` and `minWidth` values of the specified resistor type.
Multiplication Factor	Virtuoso XL supports the use of the multiplication factor (m-factor) as a parameter to define a one-to-many parallel relationship between a device in a schematic and multiple instances of the device in the layout. This parameter is intended for use at the schematic level.

spcsimple_nmos, spcsimple_pmos

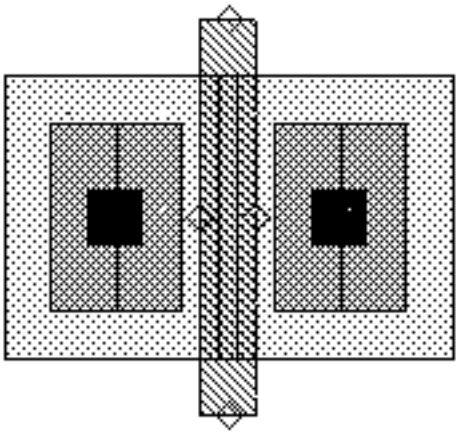
The spcsimple_nmos and spcsimple_pmos devices are simpler implementations of transistors than the spcnmos and spcpmos devices.

These simpler devices do not support wells and substrate contacts, and the contact choices are more limited. The contact coverage and the choice to have contacts are merged into a single contact coverage parameter, which is set in a cyclic field.

The splitting variable in the spcnmos and spcpmos devices that later prompts you for the number of fingers was replaced with a single parameter in the spcsimple_nmos and spcsimple_pmos devices. Any value larger than `1` in the *Number of Fingers* field splits the device. There is no choice of types of splitting. Note the stretch handles that let you graphically scale the device.

These devices support relative object design (ROD) commands, Virtuoso XL permutable pins and device abutment, and pin connection modeling (strong, weak, and must-join) for the Virtuoso custom router. These devices are not compatible with the Virtuoso compactor or the Virtuoso layout synthesizer.

The spcsimple_nmos Device



Parameters

Gate Width	Controls the width of the gate. Valid Values: a number in meters, in engineering notation, greater than or equal to the value of `minMosWidth` Example: `4e-6` Default: the value of `minMosWidth`
Gate Length	Controls the length of the gate. Valid Values: a number in meters, in engineering notation, greater than or equal to the value of `minWidth poly` Example: `1e-6` Default: the value of `minWidth poly`
Multiplication Factor(m)	Virtuoso XL supports the use of the multiplication factor (m-factor) as a parameter to define a one-to-many parallel relationship between a device in a schematic and multiple instances of the device in the layout. This parameter is intended for use at the schematic level; however, if you change the m-factor of a layout view, the system multiplies the width of the device by the specified factor. Valid Values: a positive floating-point number Default: `1`
Number of Fingers	Controls the number of fingers in the transistor. Valid Values: a positive integer Default: `1`
Threshold Value	Specifies the maximum width for each gate when *Number of Fingers* is greater than 1. Valid Values: a number in meters, in engineering notation, greater than the value of `minMosWidth * 10` Example: `30` Default: `0.0`
Left Contact Position	Positions the contacts vertically. When the contacts are not spread out evenly along the width of the drain, you can shift the contacts toward the top, center, or bottom of the drain, relative to the unrotated device. Valid Values: *Center*, *Top*, or *Bottom* Default: *Center*

Left Contact Coverage
Controls the percentage of the width of the left side of the device that the contacts cover. This lets you route other metal lines across the device without connecting them to it electrically.
Valid Values: `0`, `25`, `50`, `75`, or `100`
Default: `100`

Right Contact Position
Positions the contacts vertically. When the contacts are not spread out evenly along the width of the drain, you can shift the contacts toward the top, center, or bottom of the drain, relative to the unrotated device.
Valid Values: *Center*, *Top*, or *Bottom*
Default: *Center*

Right Contact Coverage
Controls the percentage of the width of the device that the right side contacts cover. This lets you route other metal lines across the device without connecting them to it electrically.
Valid Values: `0`, `25`, `50`, `75`, or `100`
Default: `100`

Center Contact Position
Positions the center contact, if there is more than one finger.
Valid Values: *Center*, *Top*, or *Bottom*
Default: *Center*

Center Contact Coverage
Controls the right side contacts coverage, if there is more than one finger. This lets you route other metal lines across the device without connecting them to it electrically.
Valid Values: `0`, `25`, `50`, `75`, or `100`
Default: `100`

A

Building Pcells with ROD Functions

The Pcells described in this reference were built in part with relative object design functions (ROD), a set of high-level Cadence® SKILL language functions for defining simple and complex layout objects and their relationships to each other. The following code examples demonstrate how to use ROD functions to build Pcells.

Note: If you want to make changes to the code in any of the Pcell devices, utility functions, or installation files, you should create a copy of the original sample file using a new filename and make your changes to the new file. This method preserves the sample files for future use. If you do not create new files when you customize, your custom files replace the original sample files, and when the sample Pcells are loaded again, your custom files are overwritten by the original sample files.

Multipart Paths, Subpaths, and Subrectangles

This code example utilizes the ROD function `rodCreatePath` to create a guardring consisting of an enclosure subpath and a subrectangle.

```
procedure(buildGuardRing(cv layer1 layer2 layer3 width1 width3
    length3 pathpnts terminal pinlabel enc offset chop)

; pinlabel is a boolean that determines whether a label will be
    created

    let(( guardRing )
    guardRing = rodCreatePath(
        ?cvId cv
        ?layer layer1
        ?width width1
        ?pts pathpnts
        ?encSubPath list( list(
            ?layer layer2
            ?enclosure enc
            ?choppable chop
            ?pin t
            ?termName terminal
            ?pinLabel pinlabel
            ?pinLabelHeight length3
            ?pinLabelLayer "text"
        ))
        ?subRect list( list(
```

```
            ?layer layer3
            ?width width3
            ?length length3
            ?endOffset offset
        ))
    )
))
```

Path Stitching with ROD Functions

This code example uses the ROD functions rodAlign and rodCreateRect to build a Pcell that can be used for path stitching. It is set up for double contacts.

To use this example,

1. Copy the sample `mpu.tf` from

   your_install_dir`/tools/dfII/samples/techfiles/mpu.tf`

   to the location where you store your libraries.

2. Comment out `M1_POLY1` from the `device` class of the technology file.

   This is necessary because path stitching will see two contact definitions for the same layer and choose the single-contact definition

3. Create a new technology library using `mpu.tf` as the ASCII technology file.

4. Load this file into your new library.

You can now use double contacts for path stitching.

```
; Path Stitching Example
devices(
tcCreateDeviceClass( "symbolic" "syEnhContact"
    ; class parameters
    ( (viaLayer "cont")
      (viaPurpose "drawing")
      (layer1 "poly1")
      (purpose1 "drawing")
      (layer2 "metal1")
      (purpose2 "drawing")
      (encByLayer1 .4)
      (encByLayer2 .6)
    )
    ; formal parameters
    ( (w 1.8)
      (l .6)
      (row 1)
      (column 1)
      (xPitch 2.4)
      (yPitch 1.2)
      (xBias "center")
      (yBias "center")
```

```
    )
    cw = .6
    cs = .6
    lay1 = rodCreateRect(
        ?layer  list(layer1 purpose1)
        ?width   2 * encByLayer1 + column * w + (column - 1) * cs
        ?length  2 * encByLayer1 + row * l + (row - 1) * cs
        ?pin t
        ?termName "m1p"
    )
    lay2 = rodCreateRect(
        ?layer list(layer2 purpose2)
        ?width   2 * encByLayer2 + column * w + (column - 1) * cs
        ?length  2 * encByLayer2 + row * l + (row - 1) * cs
        ?pin t
        ?termName "m1p"
    )
    for(i 0 row - 1
      yStep = i * yPitch
      for(j 0 column - 1
        contlay1 = rodCreateRect(
                ?layer  list(viaLayer viaPurpose)
                ?width cw
                ?length cw
        )
        contlay2 = rodCreateRect(
                ?layer  list(viaLayer viaPurpose)
                ?width cw
                ?length cw
        )
        rodAlign(
                ?alignObj contlay1
                ?alignHandle "cC"
                ?refObj lay1
                ?refHandle "lL"
                ?xSep encByLayer1 + cw/2 + j * xPitch
                ?ySep encByLayer1 + cw/2 + yStep
        )
        rodAlign(
                ?alignObj contlay2
                ?alignHandle "cC"
                ?refObj lay1
                ?refHandle "lL"
                ?xSep encByLayer1 + cw/2 + j * xPitch + yPitch
                ?ySep encByLayer1 + cw/2 + yStep
        )
      )
    )
    rodAlign(
       ?alignObj lay1
       ?alignHandle "centerCenter"
       ?refObj lay2
       ?refHandle "cC"
    )
)
tfcDefineDeviceClassProp(
; (viewName        devClassName      propName          propValue)
  (symbolic        syEnhContact      function          "contact")
)
tcDeclareDevice( "symbolic" "syEnhContact" "M1_PLY"
    ( (viaLayer "cont") (viaPurpose "drawing")
```

```
      (layer1 "poly1") (purpose1 "drawing")
      (layer2 "metal1") (purpose2 "drawing")
      (encByLayer1 .4) (encByLayer2 .6))
    (
      (w 1.8) (l .6)
      (row 1) (column 1)
      (xPitch 2.4)
      (yPitch 1.2)
      (xBias "center")
      (yBias "center")
    )
)
)
```