Agilent Technologies

Advanced Design System 2011.01

Feburary 2011
Cadence Library Integration

Errata The ADS product may contain references to "HP" or "HPEESOF" such as in file names and directory names. The business entity formerly known as "HP EEsof" is now part of Agilent Technologies and is known as "Agilent EEsof". To avoid broken functionality and

About Cadence Library Integration

The information provided in this document pertains to both the Agilent Technologies *RFIC Dynamic Link* and *RF Design Environment* (RFDE). These two products are provided for customers who want to take advantage of Agilent's powerful RF simulation technology while still maintaining access to the tools provided by Cadence Design Systems.

RFIC Dynamic Link

The *RFIC Dynamic Link* enables you to simulate your Cadence designs in the *Advanced Design System* (ADS) environment. Designs entered in the Cadence Schematic and stored in the Cadence design database are represented on the ADS schematic via its symbol view. The circuits can be simulated together with arbitrary combinations of ADS system and circuit components using all the circuit simulators available in ADS.

RF Design Environment

The *RF Design Environment* provides a more tightly integrated EDA solution that enables RF/mixed-signal IC designers to simulate their designs directly in the Cadence environment using the *ADSsim* RF simulator. This enables the RF/MS IC customer to take advantage of complementary features provided by both Agilent Technologies and Cadence Design Systems.

RF Design Environment Supported Platforms

If you are creating user-compiled models in ADS for use with RFDE, Supported UNIX Platforms for RF Design Environment and Supported Cadence Platforms for RF Design Environment show a history of the supported RFDE platforms.

Supported UNIX Platforms for RF Design Environment

RFDE	HP-UX	Solaris	AIX	Linux
2002C	HP-UX 11.0	Solaris 7 & 8	AIX 4.3	
2003A	HP-UX 11.0 & 11i	Solaris 7, 8, & 9	x	
2003C	HP-UX 11.0 & 11i	Solaris 7, 8, & 9	x	Red Hat 7.2, & 7.3
2004A	HP-UX 11.0 & 11i	Solaris 8 & 9	x	Red Hat 7.2, 7.3, & 8
2005A	HP-UX 11.0 & 11i	Solaris 8, 9, & 10	x	Red Hat 7.2, 7.3, & 8RHE 2.1 & 3.0
2006A32-Bit OS†	HP-UX 11.0 & 11i	Solaris 8, 9, & 10	x	Red Hat WS 3.x, WS 4.x, Novell SUSE Linux Enterprise Server 9.3
2006A64-bit OS†	Not Supported	Solaris 8, 9, & 10 with 64-bit support turned on	x	Red Hat WS 3.x, 4.0, Novell SUSE Linux Enterprise Server 9.3 compatible architectures (64-bit AMD Optern and Intel EM64T processors)
† For more detailed information on supported platforms for the current release, refer to your UNIX and Linux Installation documentation.				

Supported Cadence Platforms for RF Design Environment

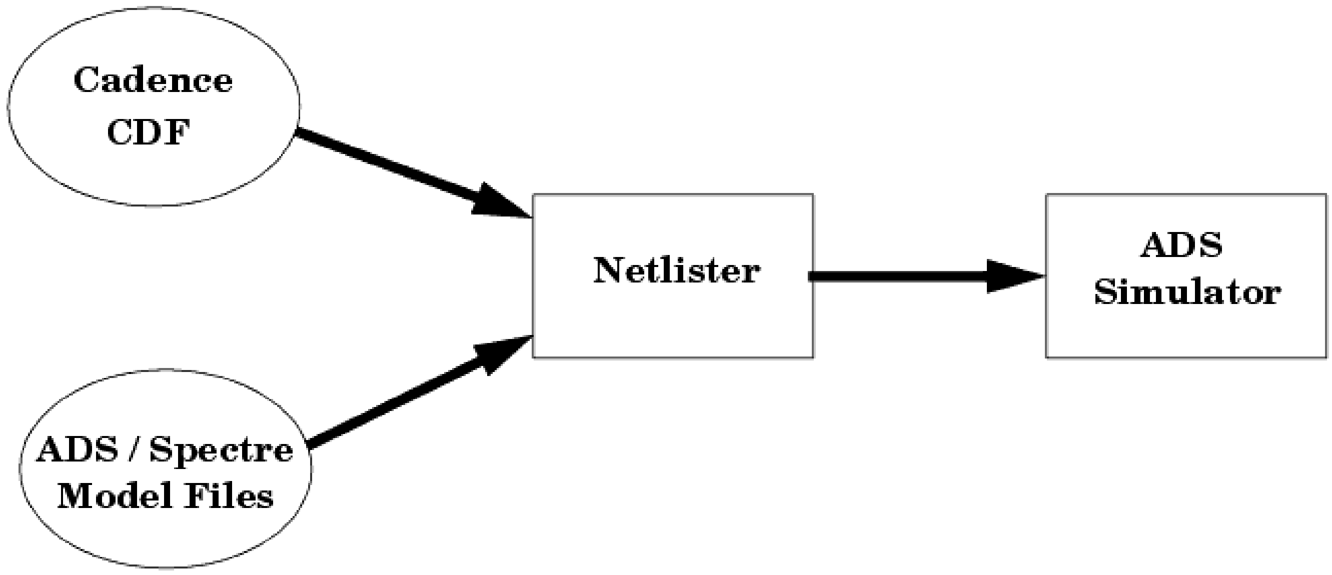
RFDE	Cadence
2002C	IC 4.4.5 QSR3 & IC 4.4.6 MSR7
2003A	IC 4.4.5 QSR4, IC 4.4.6 MSR8, & IC 5.0 MSR3
2003C	IC 4.4.6 MSR8 & IC 5.0 MSR3
2004A	IC 5.0.33 USR2 (CDBA)ICOA 5.0.33 USR2 (OpenAccess)IC 5.1.41 (CDBA)ICOA 5.1.41 (OpenAccess)
2005A	IC 5.1.41 (CDBA)ICOA 5.1.41 (OpenAccess)
2006A (32-bit Environment) †	IC 5.1.41 (CDBA)ICOA 5.1.41 (OpenAccess)
2006A (64-bit Environment) †	IC 5.1.41 (CDBA)ICOA 5.1.41 (OpenAccess) (AMD 64-bit version of Redhat not supported)
2006 Update(32- and 64-bit Environments) †	IC 5.1.41 (CDBA)ICOA 5.1.41 (OpenAccess) ICOA 5.2.51 (OpenAccess 2.2)IC 6.1.0 (OpenAccess 2.2)(AMD 64-bit version of Redhat not supported)
2008(32- and 64-bit Environments) †	IC 5.1.41 (CDBA)ICOA 5.1.41 (OpenAccess) ICOA 5.2.51 (OpenAccess 2.2)IC 6.1.0 (OpenAccess 2.2)IC 6.1.1 (OpenAccess 2.2)(AMD 64-bit version of Redhat not supported)
2008 Update(32- and 64-bit Environments) †	IC 5.1.41 (CDBA)IC 6.1.0 (OpenAccess 2.2)IC 6.1.1 (OpenAccess 2.2)IC 6.1.2 (OpenAccess 2.2)(AMD 64-bit version of Redhat not supported)
† For more detailed information on supported platforms for the current release, refer to your UNIX and Linux Installation documentation.	

For more information on User-Compiled Models in RF Design Environment, refer to the section on *Loading Shared Libraries Containing User-Compiled Models* in the Advanced Design System *User-Defined Models* documentation. The ADS documentation set can be accessed from the Agilent EEsof EDA Web site is at:

http://www.agilent.com/find/eesof-docs/

Library Integration

Both *RFIC Dynamic Link* and *RF Design Environment* require an extension of the process library to support the netlister and also require the development of model files in ADS format. This additional information is used to generate netlists in ADS format as shown in *Simulation Data Flow* (adshbapp).



Simulation Data Flow

Note
If you are planning to use components from the *basic* and *analogLib* libraries in your designs, refer to *The basic Library* (dynlnklc) and *Modifying the analogLib Library* (dynlnklc) for additional information.

If you are interested in using Spectre Model Files, refer to the *Spectre Compatible Process Design Kits* documentation in *RF Design Environment* documentation set. The RFDE documentation set can be accessed from the Agilent EEsof EDA Web site is at:

http://www.agilent.com/find/eesof-docs/

This document provides information on how to make these additions, articulated into the following two categories:

- Creating the Netlist Interface: This task consists of modifying the Cadence library database by adding ADS simulation information to the Component Description Format (CDF) and creating an *ads* Cellview for each library component.
- Creating Model Files: This is done by creating ASCII text files, formatted for ADS, that contain model parameters for each of the components.

Using Examples

Each of the above tasks is described with examples. Both RFIC Dynamic Link and RF Design Environment include a modified version of the *analogLib* library installed under *$HPEESOF_DIR/cdslibs/5.1.0 or 5.1.2* which is used in the examples. If you do not have write access to this directory or do not want to overwrite it, make a copy of the directory first as follows:

```
cd $HPEESOF_DIR/cdslibs/5.1.0 (or 5.1.2)
find analogLib -depth -print | cpio -pd
```

If you make a copy of the library (recommended), ensure that you edit your *cds.lib* file to point to your own copy of *analogLib* instead of to the original installed version.

Intended Audience

The information contained in this documentation applies to EDA engineers and managers responsible for creating and maintaining process libraries who:

- would like to implement a design flow based on the integration of ADS and Cadence DFII using the RFIC Dynamic Link or RF Design Environment.
- have an existing Cadence component library which supports at least one commercially available SPICE simulator.
- are familiar with the Cadence library structure and Component Description Format (CDF).

If you are familiar with the topics above, you can successfully complete the library modification using the information contained in this documentation.

The following rules apply to this guide

- Wherever a shell variable is set, the Korn shell syntax is presented.
- Unless otherwise mentioned, assume case sensitivity.
- If you don't understand a particular term or acronym, refer to the Glossary in the *RFIC Dynamic Link* documentation.
- For information on the RFIC Dynamic Link menus, refer to *Command Reference for RFIC Dynamic Link* (dynlnkug).

Getting ADS Device Parameter Information

This section describes how to obtain parameter information for devices supported by the Advanced Design System Analog/RF Simulator (*ADSsim*). The parameter information is required to complete the tasks outlined in subsequent sections.

The ADS Simulator provides helpful information on netlist and model formatting via a terminal window. To use the ADS Simulator for this purpose, ensure that your environment has been configured for use with RFIC Dynamic Link.

Listing Available Devices

This section describes how to use the adssim and hpeesofsim commands to list available devices.

The adssim Command

Previously, the *hpeesofsim* command was the preferred method for listing available devices. However, you can now use the alternate *adssim* command in place of the *hpeesofsim* command. The adssim command automatically sources the *bootscript.sh* file (see below) and then calls the hpeesofsim command. The adssim command also returns the netlist format for models. For example, using the adssim -help command with MSUB :

```
$ adssim -help MSUB
Microstrip Substrate Parameter Definition.
ModelName [:Name] <parameter=value> ... ; (device)
model ModelName MSUB <parameter=value> ...
model Parameters:
   Er                smorr Relative dielectric constant.
   Mur               smorr Relative permeability.
   H (m)             smorr Substrate thickness.
   Hu (m)            smorr Cover height.
   T (m)             smorr Conductor thickness.
   Cond (S/m)        smorr Conductor conductivity.
   TanD              smorr Dielectric Loss Tangent.
   Rough             smorr Conductor surface roughness.
   Secured           s---b Secured Substrate parameters.
```

The hpeesofsim Command

The hpeesofsim command uses shared libraries that are set in the $HPEESOF_DIR/bin/bootscript.sh script. Before attempting to use the hpeesofsim command, you should source the *bootscript.sh* file using one of the following commands:

. $HPEESOF_DIR/bin/bootscript.sh	(If using the Korn shell)
sh	(If using the C shell)
. $HPEESOF_DIR/bin/bootscript.sh	

Note
The above commands are only necessary if *SHLIB_PATH* for HP-UX, or *LD_LIBRARY_PATH* for SunOS or Linux, does not include the shared libraries required to run hpeesofsim.

In a terminal window, enter:

```
hpeesofsim -help
```

A list of *Available devices and analyses* are displayed.

Getting Device Parameters

This section describes how to use the *hpeesofsim* command to obtain parameter information for a specified device. From a terminal window, enter:

```
hpeesofsim -help < _device_name_ >
```

where <*device_name*> is derived using the procedure described in Listing Available Devices .

Note
All device names are case sensitive. Use the *hpeesofsim -help* command to verify the correct case and spelling.

Viewing Device Output

The output of the ADS Simulator help for a specific device is a generated list of instance and model information. The output can be divided into four parts; the *Instance Statement* , the *List of Instance Parameters* , the *Model Statement* and the *List of Model Parameters* .

The examples below show the simulator output for a Bipolar Junction Transistor (BJT). To view the entire list of device parameters in a terminal window, enter:

```
hpeesofsim -help BJT
```

1. Instance Statement - The first section of the output produces the netlist instance statement format for the device.
   Netlist instance statement format:
   ModelName [:Name] collector base emitter ... <parameter=value> ... ; (device)
   For more information, refer to *Instance Statements* in *Using Circuit Simulators*.
2. List of Instance Parameters - The second section contains the list of instance parameters that can be netlisted in the instance statement.

List of available instance parameters:

```
Parameters:
   Area               smorr Junction area factor.
   Region             s---i DC operating region, 0=off, 1=on, 2=rev, 3=sat.
   Temp  (C)          smorr Device operating temperature.
   Gbe  (Siemens)     ---rr Small Signal Base Emitter Conductance.
   Cbe  (F)           ---rr Small Signal Base Emitter Capacitance.
   Gb  (Siemens)      ---rr Small Signal External Base Conductance.
   Cbc  (F)           ---rr Small Signal Internal Base Collector Capacitance.
   Cbcx  (F)          ---rr Small Signal External Base Collector Capacitance.
   Ccs  (F)           ---rr Small Signal Collector to Substrate Capacitance.
   dQbe_dVbc  (F)     ---rr Small Signal Vbc To Qbe Transcapacitance.
   dIce_dVbe  (Siemens) ---rr Small Signal Forward Transconductance gm.
   dIce_dVbc  (Siemens) ---rr Small Signal Reverse Transconductance gmr.
   dIbe_dVbc  (Siemens) ---rr Small Signal Reverse Transconductance gmr.
   dIbx_dVbe  (Siemens) ---rr External Base Transconductance dIbx_dVbe.
   dIbx_dVbc  (Siemens) ---rr External Base Transconductance dIbx_dVbc.
   NPN                s---b NPN bipolar transistor.
   PNP                s---b PNP bipolar transistor.
   Mode               s---i Nonlinear spectral model on/off.
   Noise              s---b Noise generation on/off.
```

Example of an instance statement containing some instance parameters:
NPN:Q1 c b e s Area=10 Region=1

3. Model Statement - The third section contains the device model statement format:
   model ModelName BJT <parameter=value> ...
   For more information, refer to *Model Statements* in *Using Circuit Simulators*.
4. List of Model Parameters - The last section contains the model parameter information used to build the ASCII model file.

Note
The use of ellipse (...) in the following output format indicates that some of the information has not been shown for conciseness.

List of available model parameters:

```
model Parameters:
   NPN                s---b NPN bipolar transistor.
   PNP                s---b PNP bipolar transistor.
   Is  (A)            smorr Saturation current.
   Js  (A)            smorr Saturation current.
   Bf                 smorr Forward beta.
   Nf                 smorr Forward emission coefficient.
   Vaf  (V)           smorr Forward Early voltage.
   Vbf  (V)           smorr Forward Early voltage.
   ...
   wBvbe  (V)         s--rr Base-emitter reverse breakdown voltage \(warning\).
   wBvbc  (V)         s--rr Base-collector reverse breakdown voltage \(warning\).
   wVbcfwd  (V)       s--rr Base-collector forward bias \(warning\).
   wIbmax  (A)        s--rr Maximum base current \(warning\).
   wIcmax  (A)        s--rr Maximum collector current \(warning\).
   wPmax  (W)         s--rr Maximum power dissipation \(warning\).
   Approxqb           s---b use the approximation for Qb vs Early voltage.
   Lateral            s---b Lateral substrate geometry.
   Null               s---- Has no effect.
```

Example of Model Statement containing some model parameters (note the use of the backslash character):

```
model npn BJT NPN=yes Is=4.598E-16 Bf=175 Nf=0.9904 Vaf=22 Ikf=0.8 \
Ise=1.548E-14 Ne=1.703 Br=76.1 Nr=0.9952 Var=2.1 \
Ikr=0.02059 Isc=3.395E-16 Nc=1.13 Rb=8 Irb=8E-05 \
Rbm=3 Re=0.45 Rc=6 Xtb=0 Eg=1.11 Xti=3 Cje=8.7E-13 \
Vje=0.905 Mje=0.389 Cjc=3.6E-13 Vjc=0.4907 Mjc=0.2198 \
Xcjc=0.43 Tf=1e-11 Xtf=50 Vtf=test\(AAA\) Itf=0.32 Ptf=32 \
Tr=1E-09 Fc=0.6
```

In the previous definition, the parameter attributes have the following interpretation:

field 1: settable

s = settableS = settable and required

field 2: modifiable

m = modifiable

field 3: optimizable

o = optimizable

field 4: readable

r = readable

field 5: type

b = booleani = integerr = real numberc = complex numberd = device instances = character string

For more information on parameter attributes, refer to *Model Parameter Attribute Definitions*.

Attribute	Meaning	Example
settable	Can be defined in the instance or model statement. Most parameters are settable, there are a few cases where a parameter is calculated internally and could be used either in an equation or sent to the dataset via the OutVar parameter on the simulation component. The parameter must have its full address.	Gbe (Small signal Base-Emitter Conductance) in the BJT model can be sent to the dataset by setting `OutVar="MySubCkt.X1.Gbe"` on the simulation component.
required	Has no default value; must be set to some value, otherwise the simulator will return an error.	
modifiable	The parameter value can be swept in simulation.	
optimizable	The parameter value can be optimized.	
readable	Can be queried for value in simulation using the OutVar parameter. See settable.	
boolean	Valid values are 1, 0, yes, no, True, and False.	
integer	The maximum value allowed for an integer type is 32767, values between 32767 and 2147483646 are still valid, but will be netlisted as real numbers. In some cases the value of a parameter is restricted to a certain number of legal values.	The Region parameter in the BJT model is defined as integer but the only valid values are 0, 1, 2, and 3.
real number	The maximum value allowed is 1.79769313486231e308+.	
complex number	The maximum value allowed for the real and imaginary parts is 1.79769313486231e308+.	
device instance	The parameter value must be set to the name of one of the instances present in the circuit.	The mutual inductance component (Mutual), where the parameters Inductor1 and Inductor2 are defined by instance names of inductors present in the circuit or by a variable pointing to the instance names. Inductor1="L1" or Inductor1=Xyz where Xyz="L1"
character string	Used typically for file names. Must be in double quotes.	Filename="MyFileName"

Creating the Netlist Interface

This chapter describes how to modify the Cadence library database. This includes creating a new *ads* symbol view for each library component as well as adding an ADS simulation information section to the Component Description Format (CDF). This procedure can be divided into the following tasks:

- Creating the *ads* Symbol View for a component
- Modifying the CDF for a component
    - Getting existing CDF information for a component
    - Editing the CDF File contents
    - Loading the modified CDF file
- Modifying the component netlisting function(s)

> **Note**
> While the procedure for modifying the *analogLib npn* component is described, this same procedure can be applied to most any library component.

Schematic netlisting for the ADS Analog/RF Simulator (*ADSsim*) is achieved through custom written SKILL code that uses OASIS Direct. The netlister follows the conventions of OASIS direct, and supports the same capabilities that other OASIS Direct netlisters, such as Spectre, support.

Netlisting is provided for two alternate tool flows. The first flow involves using the Circuit Design Environment to perform simulation setup and simulation. The alternate flow involves using RFIC Dynamic Link. The code for both netlisting solutions is shared in a single context file.

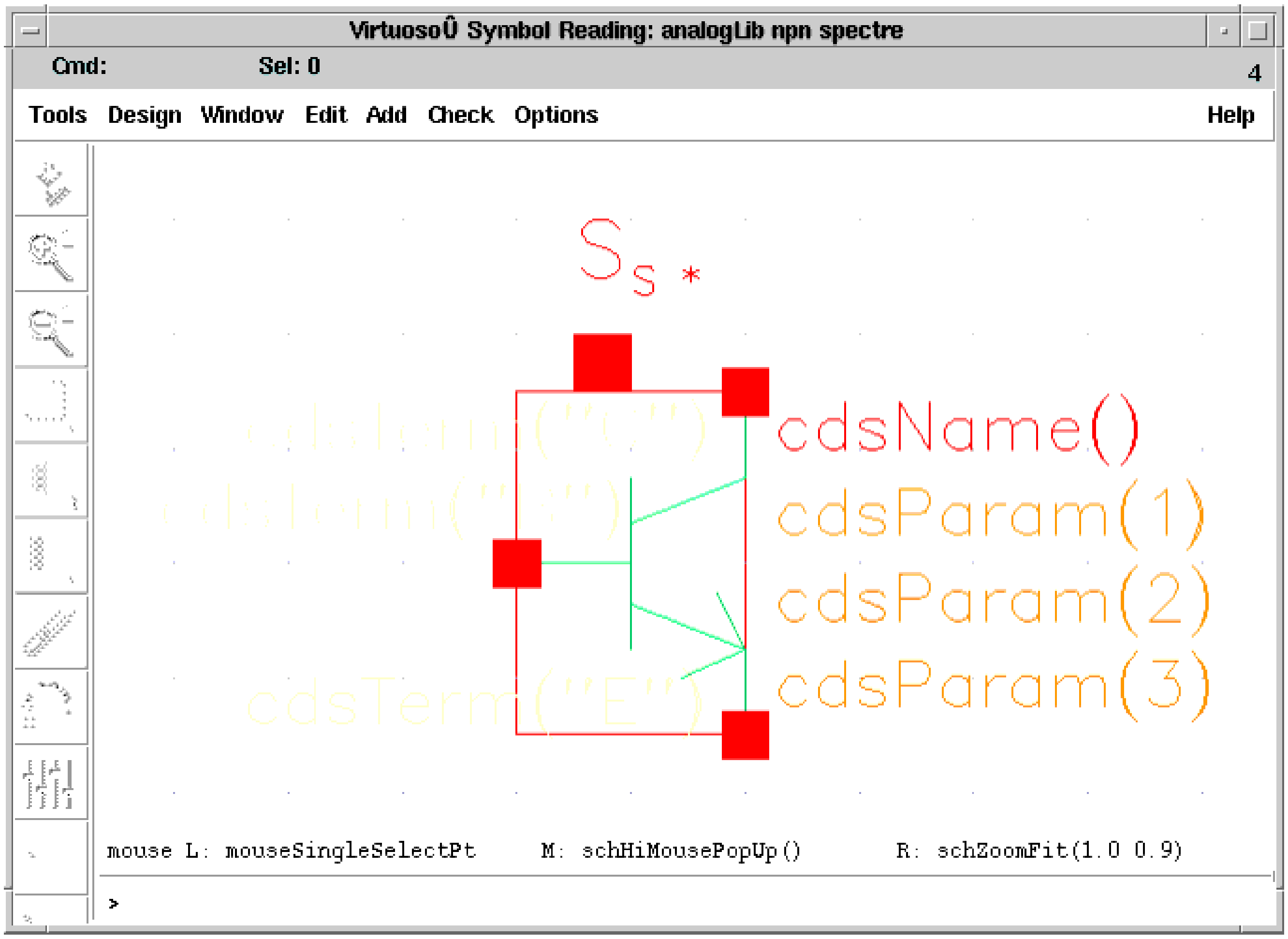
Creating the ads Symbol View for a Component

Each primitive component requires an *ads* symbol view (or *stop* view) so that the netlister knows where in the design hierarchy stops expanding the netlist. The *ads* symbol view also functions as an instance parameter template.

> **Note**
> For more detailed information on creating and modifying symbol views in Cadence, refer to *Chapter 5: Creating Symbols* in the Cadence *Virtuoso Schematic Editor User Guide* .

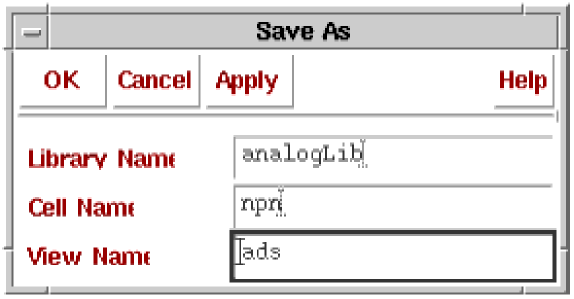
To create the *ads* view:

1. From the Cadence CIW, choose **File > Open** to open an existing symbol view (for example, the *spectre* view) of a cell such as the analogLib *npn* cell.



```
mouse L: mouseSingleSelectPt      M: schHiMousePopUp()       R: schZoomFit(1.0 0.9)
>
```

2. Choose **Design** > **Save As.** The *Save As* dialog box appears.

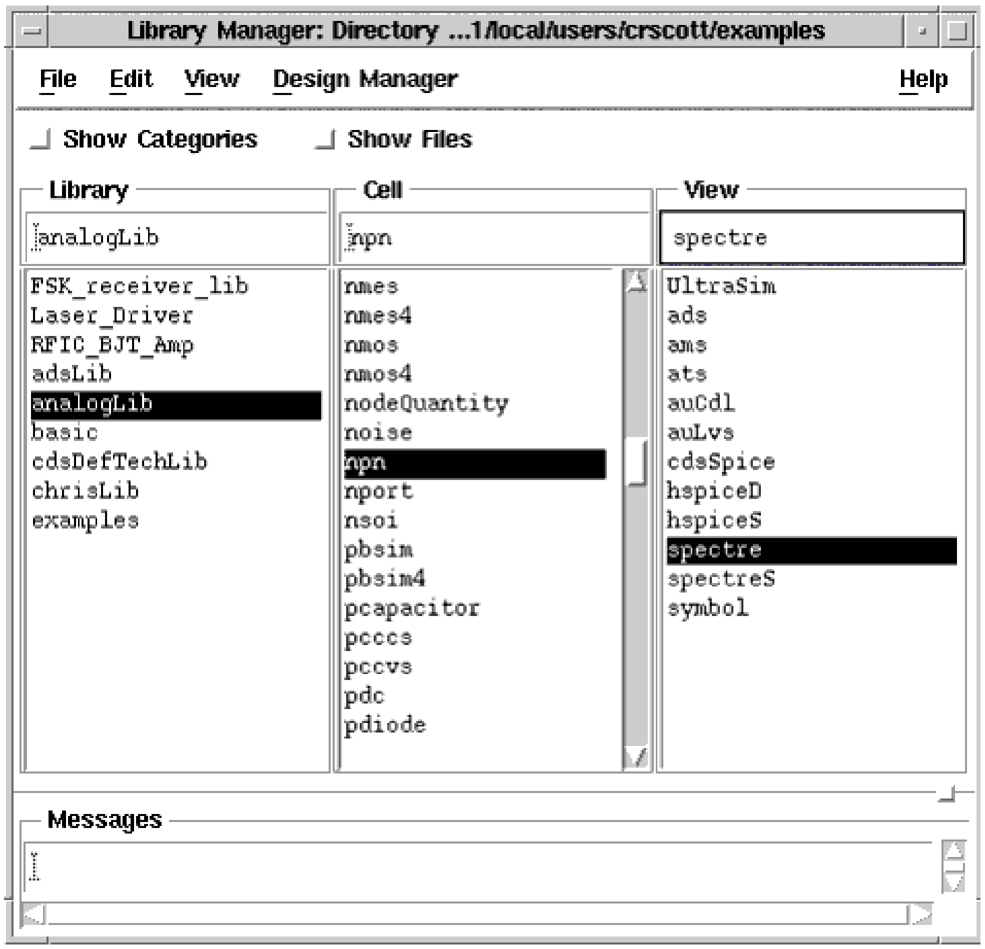


Library Name: analogLib
Cell Name: npn
View Name: ads

3. In the *Save As* dialog box, change the *View Name* field to *ads* and click **OK** . This creates the *ads* view in the analogLib database for the npn cell.

Alternatively, you can use the following procedure:

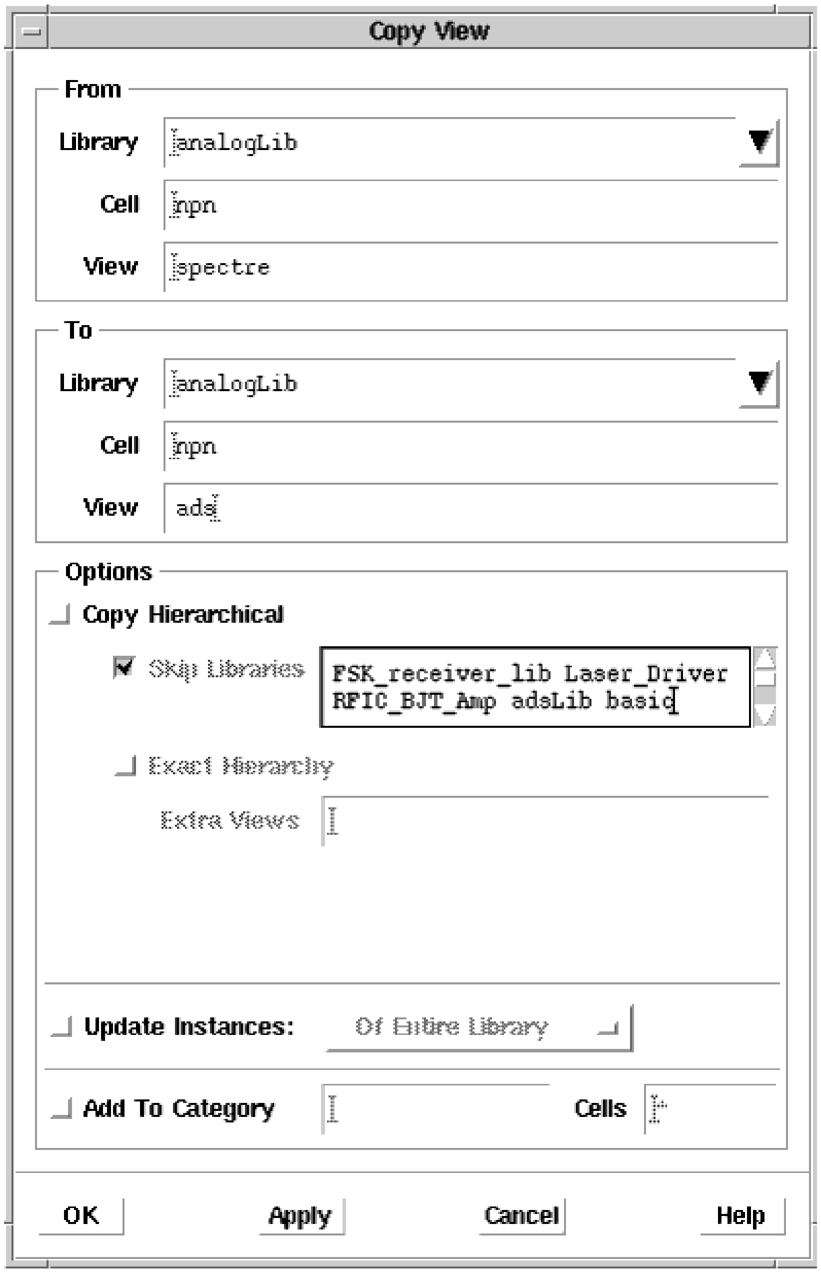
1. In the Cadence CIW, choose **Tools** > **Library Manager** . The Library Manager form appears.



Library: analogLib
Cell: npn
View: spectre

Library	Cell	View
FSK_receiver_lib	nmes	UltraSim
Laser_Driver	nmes4	ads
RFIC_BJT_Amp	nmos	ams
adsLib	nmos4	ats
analogLib	nodeQuantity	auCdl
basic	noise	auLvs
cdsDefTechLib	npn	cdsSpice
chrisLib	nport	hspiceD
examples	nsoi	hspiceS
	pbsim	spectre
	pbsim4	spectreS
	pcapacitor	symbol
	pcccs	
	pccvs	
	pdc	
	pdiode	

2. In the *Library Manager* form, choose **Edit** > **Copy** . The *Copy View* form appears.



From
Library: analogLib
Cell: npn
View: spectre

To
Library: analogLib
Cell: npn
View: ads

Options
Copy Hierarchical
Skip Libraries [✓]: FSK_receiver_lib Laser_Driver RFIC_BJT_Amp adsLib basic
Exact Hierarchy
Extra Views
Update Instances: Of Entire Library
Add To Category
Cells: *

3. In the *To* section of the *Copy View* dialog box, enter *ads* in the *View* field. Ensure that all other pertinent information is correct, then click **OK** .

Modifying the Component Description Format

To modify the Component Description Format (CDF) information for a particular library component, you need the following information:

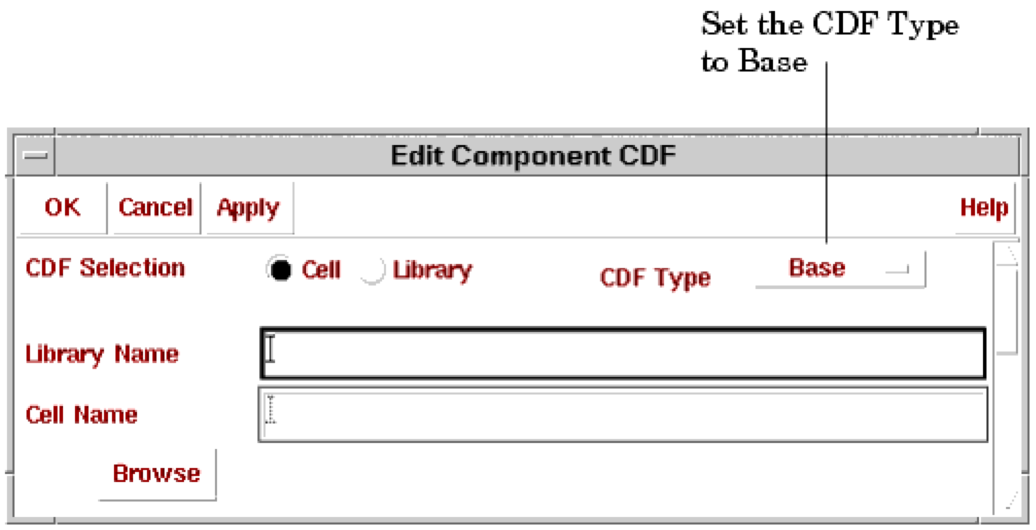
- A list of ADS instance parameters for the component. For more information, refer to *Getting Device Parameters* (dynlnklc).
- The existing CDF information for the component.

Using the CDF Editor

One method for editing the component CDF is by using the CDF editor.

To launch the CDF editor from the CIW:

1. Choose **Tools > CDF > Edit** . The Edit Component CDF form appears.

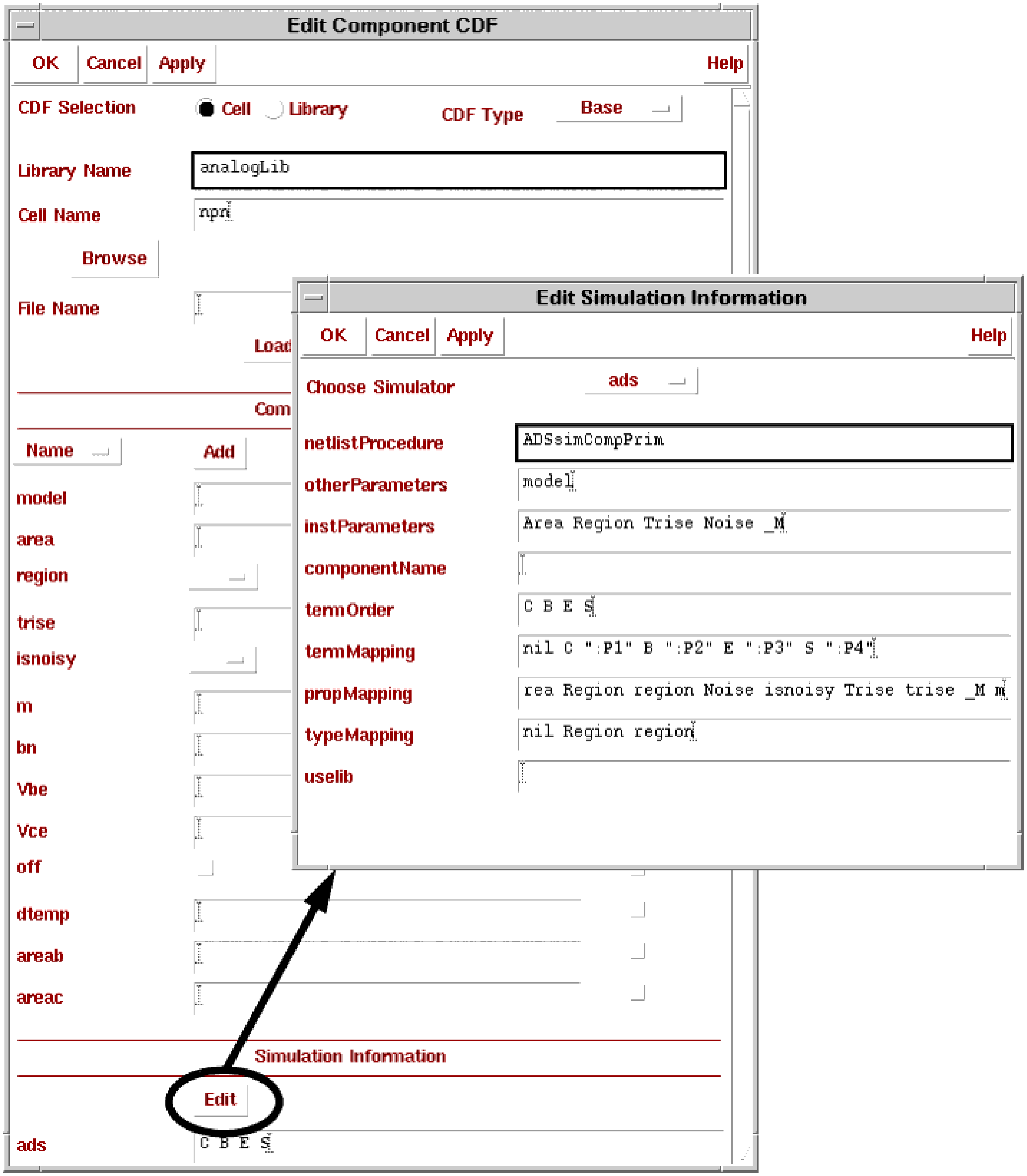


CDF Selection: [✓ Cell] Library
CDF Type: Base
Library Name
Cell Name

2. Ensure the *CDF Type* is set to **Base** .

Important
To save CDF Edit dialog box changes, you must edit the base-level CDF and have write permission to the library. For more information on CDF Type, refer to your Cadence documentation.

3. Select the component you wish to edit using the **Browse** button, or by typing it in manually.
4. To set up the simulation information for the ads tool, click the **Edit** button in the *Simulation Information* section of the Edit Component CDF form. The *Edit Simulation Information* form appears.



CDF Selection	[✓ Cell] Library
CDF Type	Base
Library Name	analogLib
Cell Name	npn
File Name	
model	
area	
region	
trise	
isnoisy	
m	
bn	
Vbe	
Vce	
off	
dtemp	
areab	
areac	
ads	C B E S

Choose Simulator	ads
netlistProcedure	ADSsimCompPrim
otherParameters	model
instParameters	Area Region Trise Noise _M
componentName	
termOrder	C B E S
termMapping	nil C ":P1" B ":P2" E ":P3" S ":P4"
propMapping	rea Region region Noise isnoisy Trise trise _M m
typeMapping	nil Region region
uselib	

Cadence Edit Component CDF and Edit Simulation Information Form

Note
While the CDF *Edit Simulation Information* form may be used to edit the CDF, it is more useful to verify what is in the CDF database. Using *cdfDump()* and a text editor is more reliable for editing the CDF.

Using cdfDump

In addition to using the Edit Component CDF form, it is also possible to create a SKILL representation of a components CDF. This SKILL representation can be exported to a file, edited, and then reloaded in order to change the CDF of a component.

The to use the cdfDump command:

1. Enter the command cdfDump(" *<library>* " " *<file>* " ?cellName " *<cell>* ") in the Cadence CIW. This will create the SKILL CDF file. For example:
   cdfDump("analogLib" "/tmp/npn.cdf" ?cellName "npn")
2. Edit the CDF information (see *Cadence Component Description Format User's Guide)* in a text file to make modifications. For example:
   vi /tmp/npn.cdf
3. Find the section that begins with "cdfId->simInfo->ads='(nil".
4. Edit the fields as appropriate (see Adding CDF Simulation Information for ADS). The CDF file consists of two main parts. The first part defines the generic *parameters* used, for example, *width* and *length* . These parameter definitions are shared by all the supported simulators under the Affirma Analog Circuit Design Environment. The second part, known as the simulation information (*simInfo*) section, details how some subset of these parameters apply to each different simulator. This section determines how each component instance is netlisted and how its model arguments and model parameter values are output in the netlist. The simInfo sub-section of primary interest here is the *ads* siminfo sub-section, which needs to be created in order for the component to be supported by RFIC Dynamic Link.
5. Save the modified file.
6. To update the component, load the modified CDF file.

Example CDF File

The actual CDF file may resemble the following. For conciseness, only a few of the CDF parameter definitions and siminfo sub-sections have been shown here. This file was obtained as outlined in the previous step. The *ads Simulation Information* sub-section is shown highlighted.

```
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
LIBRARY = "analogLib"
CELL    = "npn"
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
let( ( libId cellId cdfId )
   unless( cellId = ddGetObj( LIBRARY CELL )
       error( "Could not get cell %s." CELL )
   )
   when( cdfId = cdfGetBaseCellCDF( cellId )
       cdfDeleteCDF( cdfId )
   )
   cdfId  = cdfCreateBaseCellCDF( cellId )
   ;;; Parameters
   cdfCreateParam( cdfId
       ?name           "model"
       ?prompt         "Model name"
       ?defValue       ""
       ?type           "string"
       ?display        "artParameterInToolDisplay('model)"
       ?parseAsCEL     "yes"
...
;;; Simulator Information
   cdfId->simInfo = list( nil )
   cdfId->simInfo->ads = '( nil
       namePrefix          ""
       netlistProcedure    ADSsimCompPrim
       otherParameters     (model)
       instParameters      (Area Region Trise Noise _M)
       termOrder           (C B E S)
       termMapping         (nil C ": P1" B ": P2" E ": P3" S ": P4")
       propMapping         (nil Area area Region region Noise isnoisy
                          Trise trise _M m)
       typeMapping         (nil Region region)
       uselib              nil
   )
...
```

Adding CDF Simulation Information for ADS

A detailed explanation of the CDF information fields is provided in your Cadence documentation. However, in addition, the following information applies to RFIC Dynamic Link.

There are 9 fields that you can specify for the ADS Simulation Information definition. It is not necessary to define values for all fields. Empty fields are ignored, or will use a default value as a specified in the description of each field.

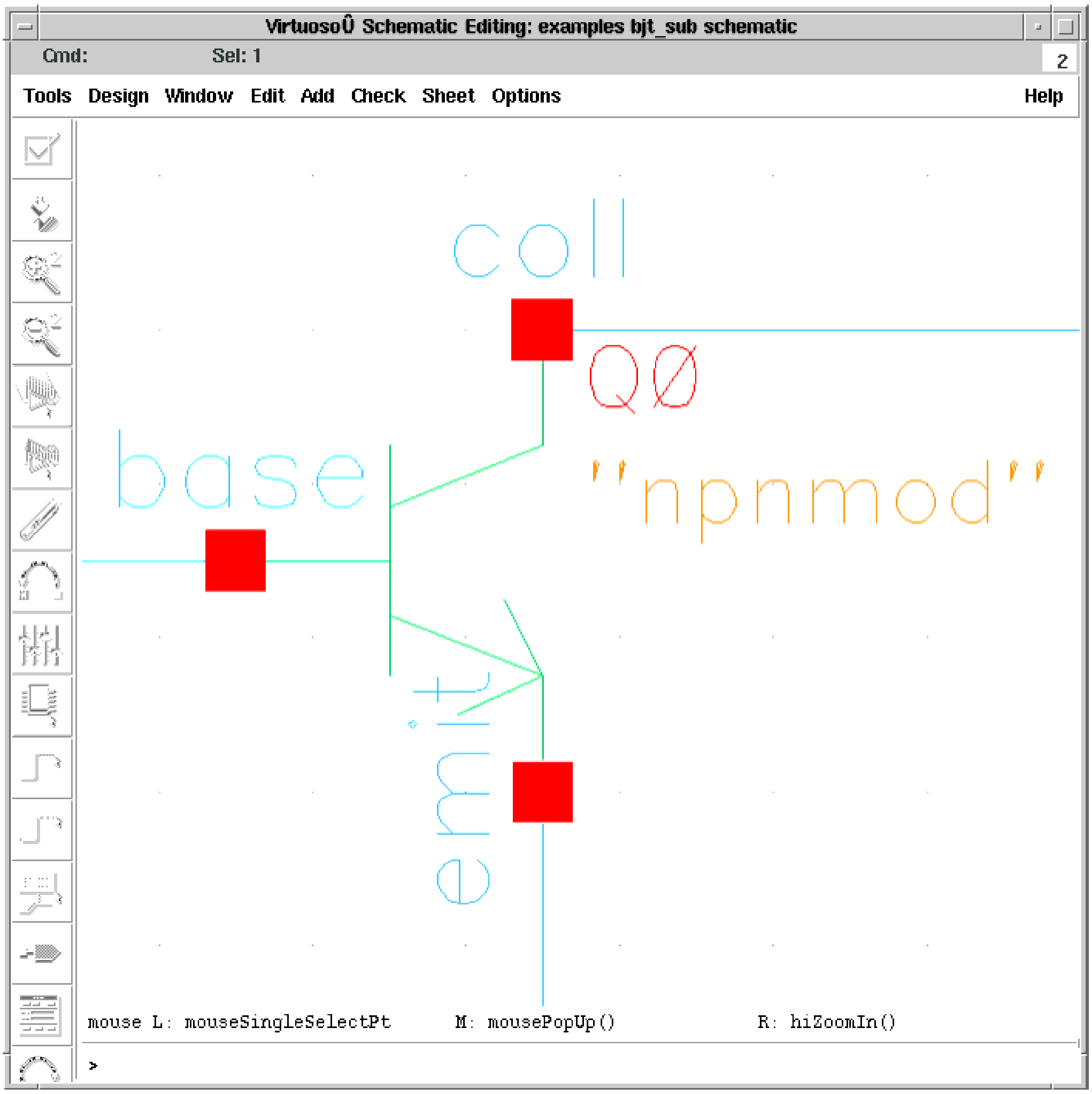
- netlistProcedure: This field is used to specify the name of the SKILL procedure to use to netlist the component. Use the built-in netlisting functions *ADSsimCompPrim* for devices and *ADSsimSubcktCall* for subcircuits. If it is left blank, it will default to *ADSsimSubcktCall* .
- otherParameters: The otherParameters field has no effect on a parameter that does not use the *artParameterInToolDisplay* function to determine whether a parameter is displayed or not. For example, the most commonly specified parameter in otherParameters is *model* . In the ADS simulator, model determines the component type of the instance. It is not netlisted in the parameter list, but it is necessary to enable you to see and edit its value. For an npn component, it is put in the otherParameters list, because, for an npn, the display value for model is determined by *artParameterInToolDisplay('model)* . If model does not exist in otherParameters, and the tool filter is set to ADS, model will not be displayed in the edit properties dialog.
- instParameters: This field is used to specify the parameters that will be output for an instance. The instParameters field also defines the parameters that will be output for

hierarchical parameter passing in the subcircuit definition of hierarchical schematics. The instParameters field should use the name as it needs to appear in the netlist. This may be different from the actual name of the parameter. Parameter name mapping is handled in the *propMapping* field. The order of the parameters is unimportant for ADS. For the ADS simulator, if the value of the parameter is empty, it will not be output into the netlist. As an example, the npn has parameters of *area* , *region* , *trise* , *noise,* and *m* . The instParameters field for ADS is set to the list (Area Region Trise Noise *M). None of these names actually represent the CDF parameter name, they are all mapped parameter names (see _propMapping*).

- componentName: This field is used to define the simulator primitive, model, or subcircuit for an instance. The value is derived first by checking to see if there is a parameter named model. If there is, and the value of model is not empty, the value of the model parameter is used as the component name. If model is empty, or the parameter is not defined in the CDF, the value of componentName is consulted and used. If componentName is empty, the name of the cell is used as the component name.
- termOrder: This field specifies the order in which the terminals are netlisted. This information is obtained for each ADS component by entering:
  hpeesofsim -help < *device_name* >
  For subcircuits, the termOrder field also specifies the hierarchical terminal order that is created in the subcircuit definition statement. The pin names can be specified as symbols or as strings. If bus notation is specified, the bus vector element will be expanded completely prior to outputting the next specified terminal.
- termMapping: This field is used to map the name of an instance pin to the simulator name that is output in the PSF file. The termMapping field must be defined for DC Current annotation to function properly. For the ADS simulator, the terminal mapping should be specified as the terminal number specified in the termOrder field, prefixed with *P* . In order to get the negative value of a bi-directional pin, prefix the number with *minus.P* . Each terminal should have a ":" separator specified, and should be specified as a string. The value should be specified as a disembodied list. It is also important to specify the name of the terminal exactly as it appears in the *termOrder* field. If the termOrder field is specified using strings, use strings in the termMapping field. If the termOrder field is specified using symbols as the pin names, use symbols in the termMapping field. For example, the npn has a termOrder of '(C B E S). The *C* terminal is the first in the list, and thus has the terminal number *1* as far as the ADS simulator is concerned. The proper terminal mapping for the simulator is ": P1 " . The complete terminal mapping is the list '(nil C ": P1" B ": P2" E ": P3" S ": P4").
- propMapping: This field is used to specify how CDF property names should have their names mapped to netlisting parameter names. It is a disembodied list. The mapped name should be specified, followed by the CDF property name. Note that mapping a property to *model* will allow that parameter's value to be used as the componentName. For example, the npn has Area as one of the parameters specified in the instParameters field. The actual CDF property name for Area is *area* . The area parameter needs to be mapped to Area for the ads simulator. This is done by setting propMapping to '(nil Area area). The npn actually has several parameters that need to be mapped, so more can be added to the list. The final propMapping for the npn becomes '(nil Area area Region region Noise isnoisy Trise trise _M m).
- typeMapping: The typeMapping field enables you to map parameter values so they are output in a specific format. The typemapping field is specified as a disembodied list, where the name of the instParameter field to do value mapping on is specified, followed by the name of the mapping to perform. The name of the mapping is

actually mapped into a function name, which is called with the value of the parameter. The type map name will be prefixed with *rfdeNetlistTypemap_* to get the final name. Thus, a typeMapping of (nil L1 string) would take the value of the L1 parameter, and pass it to the function *rfdeNetlistTypemap_string* .
To add more type mappings, you must create your own skill procedures, and follow the naming process of rfdeNetlistTypemap_ <*your type map name*> . Some of the relevant type mappings shipped with the product are as follows:
string - Returns the value enclosed in double quotes if the value passed in was a string, nil otherwise.
boolean - Returns "yes" if the value passed in is "t". Otherwise, "no" is returned.
region - Returns the expected ADS integer, based on the value passed in. Expected passed in values come from the region parameters specified in cyclic fields for assorted analogLib components.

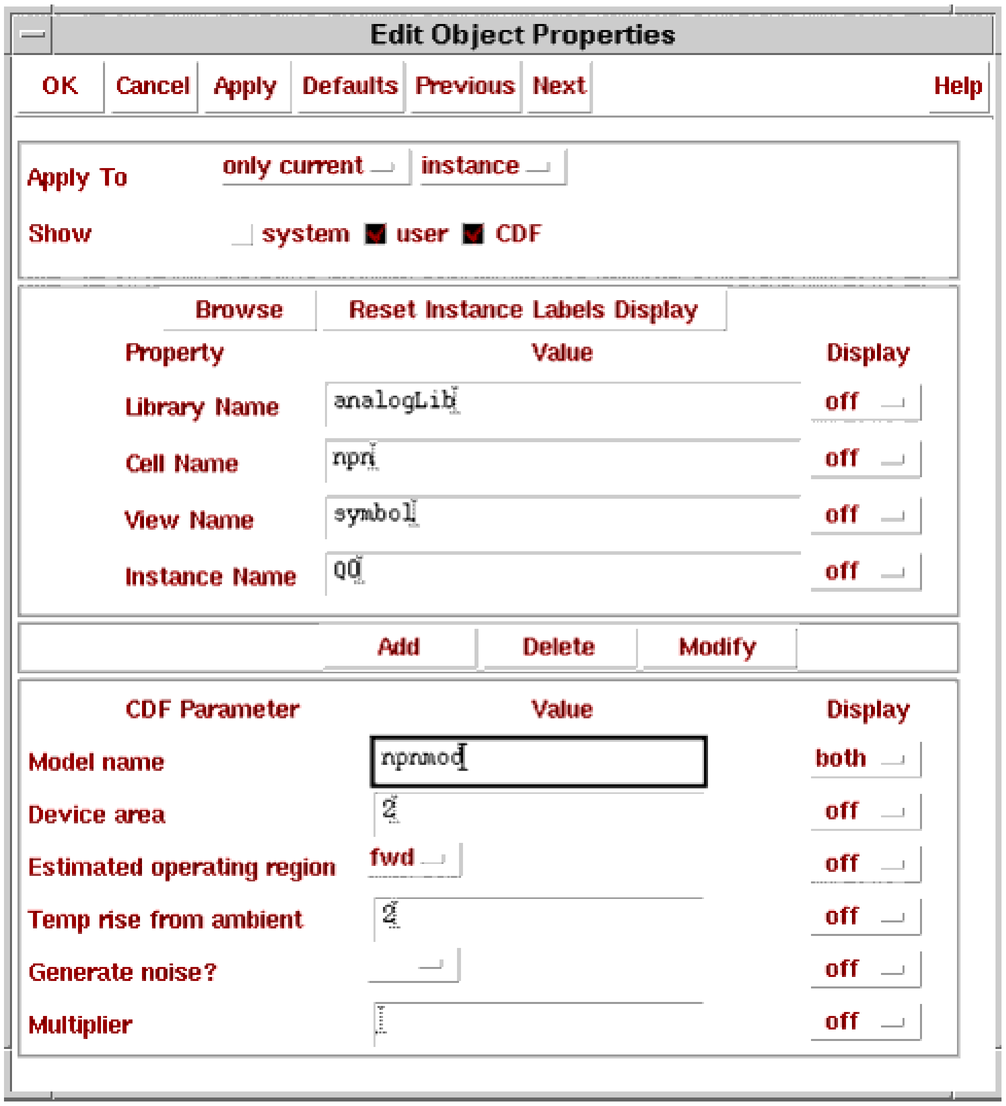
- uselib: In the ADS simulator, there are libraries of pre-defined netlist fragments. These are usually subcircuits that use elaborate equations to define parameter values. These component libraries must be accessed by preceding the netlist instance with a C Pre-processor directive, **#uselib** " <*library*> ", " <*component*> ". In this case, the component should be picked up from the *componentName* field. The library does not necessarily represent the Cadence library name, so it is necessary to specify what the simulator library name is. For example, the nport2 device in analogLib is netlisted as an S-parameter device. However, the S-parameter devices in the ADS simulator are not primitives - they are actually netlist based subcircuit fragments. In order to access the device, it is necessary to prefix the nport2 with a pre-processor directive, `#uselib "ckt", "nport2"` . This specifies that the nport2 definition can be obtained from the *ckt.library* file. The uselib library name to file name mappings are defined in the file $HPEESOF_DIR/circuit/config/ADSlibconfig.
The *npn* has been instantiated as shown in the figure below with the connecting wires named according to the device terminals.



```
mouse L: mouseSingleSelectPt        M: mousePopUp()            R: hiZoomIn()
>
```

Instance of npn Component

The object parameters for this instance have been set as follows:



Apply To: only current, instance

Show: system [✓ user] [✓ CDF]

Property	Value	Display
Library Name	analogLib	off
Cell Name	npn	off
View Name	symbol	off
Instance Name	Q0	off

CDF Parameter	Value	Display
Model name	npnmod	both
Device area	2	off
Estimated operating region	fwd	off
Temp rise from ambient	2	off
Generate noise?		off
Multiplier		off

The instance statement on the ADS netlist corresponding to this instance will appear as follows:

```
...
npnmod:Q0 coll base emit 0 Area=2.0 Region=1 Trise=2.0
...
```

The instance statement on the ADS netlist corresponding to this instance contains the following parameters:

- **npnmod** The netlister evaluates the expression contained in the *componentName* CDF field and in this case picks up the value of the model name property.
- **Q0** The instance name value is defined in the *Instance Name* field of the Edit Object Properties form.

  Note
  Illegal characters are mapped to legal characters.

- **coll base emit 0** The first three entries are taken from the names of the nodes to which the device is attached (see Instance of npn Component). In this case, the names have been explicitly assigned but the same applies to system generated node names. The *termOrder* field in the CDF controls the order in which the terminals are netlisted.

Note
The *progn* SKILL function is no longer supported by RFIC Dynamic Link in Cadence version 4.4.5 and above.

- **Area=2.0 Region=1 Trise=2.0** The parameters *Area* , *Region* and *Trise* are listed in the *instParameters* field of the component CDF, therefore they are netlisted as instance properties if their value has been set on the instance. If the field is left blank, the parameter is not netlisted and the simulator uses the default value.

Additional Notes for Simulation Information Fields

When errors in the CDF file are loaded with load < *file* >, command errors may not be reported. If this occurs, the corresponding *ads* simulation view for the device is not created.

Loading the Modified CDF File

After modifying the CDF text file to support the ADS simulator, load the edited file from the CIW using the SKILL command, load. For example:

```
load "/tmp/npn.cdf"
```

This automatically updates the Cadence library database and saves the new CDF information in the database, provided you have write permissions.

Modifying the Component Netlisting Function(s)

Each simulator can use its own netlist function to write out a component instance in its own netlist format. By default, the *nlPrintInst* function is used in RFIC Dynamic Link to format any instance where a custom netlist instance has not been specified.

Note
The *ADSsimCompPrim, ADSsimSubcktCall, IdfDevPrim* , *IdfCompPrim* , and *IdfSubcktCall* functions were used in the past and are all still available for backward compatibility. However, these are not the recommended functions. Each of these functions simply calls the *nlPrintInst* function.

If you need to create your own functions, SKILL code for these built-in functions is provided in:

$HPEESOF_DIR/cdslibs/adsLib/netlistFuncs.il

Examples of custom netlisting functions are available in:

$HPEESOF_DIR/cdslibs/skill/analogLibFuncs.il

Setting up a Component Stop View

In OASIS direct, when an instance is encountered, the netlister must decide whether the instance represents a hierarchical schematic view, or whether it represents a simulator primitive. The ADS netlister supports schematic views and extracted views for use as hierarchical schematics. All other views should be represented as hierarchical stopping points. Having a "stop view" does this.

In the environment setup, a stop view list is specified. Traditionally, the stop view for a simulator is the same as the tool name. For the ADS simulator, this means that the stop view "ads" should be used.

Copying the symbol view to a new symbol view called *ads* will create the stop view. In order to have this view used as the stop view, the stop view list must have *ads* put into it.

It is not necessary for the stop view to be identical to the symbol view. The stop view simply has to have the pins that are specified by the termOrder field of the simInfo defined in it. In many cases, the stop view will have pins with inherited connection properties on them, so that the stop view is different from the symbol view. An inherited connection pin is a pin that will be netlisted for hierarchy, but does not get a graphical representation in the schematic editor.

Setting up a Special simInfo for a Stop View (viewInfo)

In addition to setting up a component definition for a simulator, OASIS direct also allows the specification of a special netlisting definition for a particular stop view. This feature was included in OASIS direct primarily for supporting verilogA behavioral modules, but can be used for other purposes as well. If a viewInfo is not specified, the simInfo for the tool will be used, no matter which stop view is encountered. However, if a viewInfo is defined for a particular stop view, the viewInfo definition will be used instead of the tool definition as defined in simInfo. In the case of verilogA, the stop view definition is used to determine whether a primitive based subcircuit or whether a behavioral module would be used for a spectre simulation. In Advanced Design System, this could be done in a similar manner, except that a subcircuit with a symbolically defined device (SDD) as the behavioral element might be used instead of a subcircuit made up of primitive elements.

The viewInfo is added into the CDF as a disembodied list, similar to the way that the simInfo disembodied list is added. This can only be done via SKILL code; the Edit Component CDF form does not provide a way of editing the viewInfo records. Thus, to add new viewInfo definitions, it is required to use the *cdfDump* utility.

In the viewInfo record, the key field is the name of the stop view, not the simulator. This can be a problem if two simulators want to use the same stop view, but need different simulator definitions. It is crucial to ensure that definitions defined for viewInfo's can be shared between all of the simulators that will use them.

The viewInfo uses slightly different field names from the simInfo. The valid fields for a viewInfo are shown in *Stop View Information (viewInfo) Definition*. These fields will override the values specified in the simInfo for the tool.

Parameter	**Description**
netlistProc	This field takes the place of netlistProcedure in the simInfo. The netlistProc field enables you to override the simInfo netlisting function for a particular stop view.
moduleName	This field takes the place of componentName in the simInfo. Like componentName, it will consult the value of model prior to consulting the moduleName value.
parameterList	The parameterList field takes the place of the instParameters field.
termOrder	This field will override the value specified in the simInfo termOrder field.
termMapping	This field will override the value specified in the simInfo termMapping field.
propMapping	This field will override the value specified in the simInfo propMapping field.
typeMapping	This field will override the value specified in the simInfo typeMapping field.
uselib	This field will override the value specified in the simInfo uselib field.

For more information on the simInfo definition, refer to Adding CDF Simulation Information for ADS.

Example viewInfo

```
cell=ddGetObj("analogLib" "npn")
cdf=cdfGetBaseCellCDF(cell)
cdf->viewInfo=list(nil)
cdf->viewInfo->SDD='(nil netlistProc IdfCompPrim moduleName sddNpn
parameterList (Area Region Mode Temp Noise) termOrder (C B E S) termMapping
(nil C ": P1" B ": P2" E ": P3" S ": P4") propMapping (nil Area area Region
region Mode mode Temp temp Noise isnoisy) typeMapping nil uselib nil)
cdfSave(cdf)
```

Special Functions and Properties

The tables provided in this section include information on special functions or properties that can be used for schematic netlisting.

Available Instance Netlisting Functions

Parameter	Description
ADSsimCompPrim	This is the component primitive netlisting function. It should be used for any instance that is not a hierarchical schematic. This includes components that have netlist based schematic definitions.
ADSsimSubcktCall	This is the hierarchical schematic netlisting function. It should be used for any instance that is a hierarchically defined schematic/extracted view. The schematic should be defined in the dfII environment, not as a netlist based include.

Special Functions handled by the Netlister

Parameter	Description
pPar	The pPar function specifies that a parent parameter should be used. In the ADS simulator, parent parameters are accessed as local variables. In order to get the value correctly, the parameter name specified by pPar is output directly into the netlist. The value is not replaced.
iPar	The iPar function specifies that a different instance parameter should be accessed to get its value. The iPar function will be replaced in entirety by the value designated by the iPar function.
atPar	The atPar function specifies that the value of the specified parameter should be inherited from anywhere in the hierarchy. At present, this function will replace the atPar function with the value specified.

Special Properties Interpreted by the Netlister

Property	Description
nlAction	Currently, only one value can be specified for nlAction, *ignore* . If the property nlAction=ignore is set on an instance, the instance will not be netlisted. This will leave an open in the circuit where the component would have been.
Inherited Connections	If a netset property is placed on an instance, it represents a hierarchical port. Connectivity can be inherited from higher levels of the hierarchy by specifying different values on the port. This allows for nodes that are ordinarily considered global connections to become variable for different parts of the hierarchy (e.g. VDD can be set to 3v for one part of the hierarchy, but 2.5v for a different part). The ADS netlister supports the full specification of inherited connections for the Cadence dfII schematic editor.

Creating Model Files

This section describes how to create ASCII-text process-dependent model files, formatted for the ADS Simulator. These files are stored separate from the Cadence library database, in a model library directory. The netlister will simply append the model file to the final top-level ADS netlist without a syntax check. The ADS Simulator requires the syntax of these files to be exact.

To build model files in ADS format, you'll need the following information:

- The basic built-in ADS component parameter information (refer to *Getting Device Parameters* (dynlnklc)).
- The ADS Simulator Input format information (refer to *ADS Simulator Input Syntax* in the *Using Circuit Simulators* documentation).

This section describes the following tasks:

Creating a Simple ADS Model File

Once the model parameters are known, you can create an ADS model file using an ASCII text editor. In your text editor window, type in the complete model statement in the appropriate format for the selected device as defined in part 3 of *Viewing Device Output* (dynlnklc). As you build the ADS model file, be aware of the following:

- The model statement must be on a single line. Use the backslash (see the following example) as a line continuation character.
- The instance and model parameter names are case sensitive.
- If a parameter is not specified, the ADS simulator uses a default parameter value. These values are documented in the *Introduction to Circuit Components* documentation.

Example

```
model npn BJT NPN=yes Is=4.598E-16 Bf=175 Nf=0.9904 Vaf=22 \
Ikf=0.8 Ise=1.548E-14 Ne=1.703 Br=76.1 Nr=0.9952 Var=2.1 \
Ikr=0.02059 Isc=3.395E-16 Nc=1.13 Rb=8 Irb=8E-05 \
Rbm=3 Re=0.45 Rc=6 Xtb=0 Eg=1.11 Xti=3 Cje=8.7E-13 \
Vje=0.905 Mje=0.389 Cjc=3.6E-13 Vjc=0.4907 Mjc=0.2198 \
Xcjc=0.43 Tf=1e-11 Xtf=50 Vtf=1.2 Itf=0.32 Ptf=32 \
Tr=1E-09 Fc=0.6
```

Note
Spectre model files are also supported. If you are interested in using Spectre model files, refer to the *Spectre Compatible Process Design Kits* documentation in the Components, PDKs, and Models section of the *RF Design Environment* http://www.agilent.com/find/eesof-docs _ documentation set. The RFDE documentation set can be accessed from the Agilent EEsof EDA Web site is at: http://www.agilent.com/find/eesof-docs/

Creating a Parametric Subnetwork Model File

Device models, especially for active devices, often consist of complex combinations of primitive components such as resistors, inductors, capacitors, diodes and transistors. These model files are thus structured as subnetworks, that also allow parameters to be set on the instance and passed down the hierarchy to the subnetwork.

The syntax supported by the ADS Simulator is described in *Subcircuit Definitions* in the *Using Circuit Simulators* documentation.

Example

```
define npn1 \( c b e s \)
parameters area=1 region=1 noise=1
; Calculate parasitics based on the passed in parameters
rdiff=.001
areac=area\*.25
areab=area\*.5
areae=areab\*.25
rc=areac\*1e12\*rdiff
rb=areab\*1e12\*rdiff
re=areae\*1e12\*rdiff
cs=area\*1e12\*1e-15
; Define the BJT model used by the npn instance
model NPN BJT NPN=yes Is=4.598E-16 Bf=175 Nf=0.9904 Vaf=22 Ikf=0.8 \
Ise=1.548E-14 Ne=1.703 Br=76.1 Nr=0.9952 Var=2.1 \
Ikr=0.02059 Isc=3.395E-16 Nc=1.13 Rb=8 Irb=8E-05 \
Rbm=3 Re=0.45 Rc=6 Xtb=0 Eg=1.11 Xti=3 Cje=8.7E-13 \
Vje=0.905 Mje=0.389 Cjc=3.6E-13 Vjc=0.4907 Mjc=0.2198 \
Xcjc=0.43 Tf=1e-11 Xtf=50 Vtf=1.2 Itf=0.32 Ptf=32 \
Tr=1E-09 Fc=0.6
; Instances for the subcircuit
R:rc c c1 R=rc
R:rb b b1 R=rb
R:re e e1 R=re
C:csub s 0 C=cs
NPN:qin c1 b1 e1 s Area=area Region=region Noise=noise
end npn1
```

Note
Spectre model files are also supported. If you are interested in using Spectre model files, refer to the *Spectre Compatible Process Design Kits* documentation in the Components, PDKs, and Models section of the *RF Design Environment* http://www.agilent.com/find/eesof-docs _ documentation set. The RFDE documentation set can be accessed from the Agilent EEsof EDA Web site is at: http://www.agilent.com/find/eesof-docs/

Defining Instance Parameters using Expressions

Instance parameters must be defined in the *Component Parameters* section of the Cadence CDF as described in the *Cadence Component Description Format User's Guide* . RFIC Dynamic Link and RF Design Environment support netlisting of instance parameters that contain Cadence AEL expressions, such as math operators, *iPar* , *pPar* etc.

Defining Model Parameters using Expressions

Model parameters contained in ADS model files can include expressions. The expressions can be defined by arbitrary combinations of predefined ADS functions, math operators and Boolean operators. For a list of functions and operators supported, refer to the *Simulator Expressions* documentation.

For an expression to be correctly evaluated by the ADS Simulator, both the syntax of the expression and the value of the variables used in the expression must be defined in one of the following places:

1. Directly in the model file.
2. In a separate file which is included in the top level netlist.
3. In a separate file which is included in the model file.

Note
These different methods can be used in combination, with expressions defined in different places, as long as there is a single definition for each expression.

Example

This model file for a BJT contains a model parameter, Vtf, that is defined as an expression of the variable AAA.

```
model npn BJT NPN=yes Is=4.598E-16 Bf=175 Nf=0.9904 Vaf=22 Ikf=0.8 \
Ise=1.548E-14 Ne=1.703 Br=76.1 Nr=0.9952 Var=2.1 \
Ikr=0.02059 Isc=3.395E-16 Nc=1.13 Rb=8 Irb=8E-05 \
Rbm=3 Re=0.45 Rc=6 Xtb=0 Eg=1.11 Xti=3 Cje=8.7E-13 \
Vje=0.905 Mje=0.389 Cjc=3.6E-13 Vjc=0.4907 Mjc=0.2198 \
Xcjc=0.43 Tf=1e-11 Xtf=50 Vtf=test\(AAA\) Itf=0.32 Ptf=32 \
Tr=1E-09 Fc=0.6
```

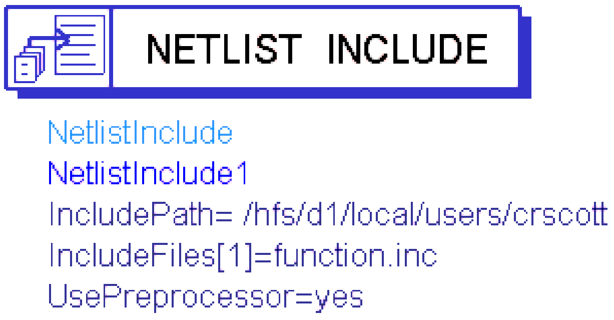
In order to simulate this model in ADS, the expression *test* needs to be defined and a value must be given to the variable *AAA* .
Assuming that:
test(x)=x*1.2
AAA=1

Do one of the following:

1. Append the definition of *test* and *AAA* to the model file:

```
model npn BJT NPN=yes Is=4.598E-16 Bf=175 Nf=0.9904 Vaf=22 Ikf=0.8 \
...
Xcjc=0.43 Tf=1e-11 Xtf=50 Vtf=test\(AAA\) Itf=0.32 Ptf=32 \
Tr=1E-09 Fc=0.6
test\(x\)=x\*1.2
AAA=1
```

2. Create a separate ASCII file (for example, *function.inc*) containing the definition of *test* and *AAA* .
If you are using RFIC Dynamic Link, place a Netlist Include Component on the top level ADS schematic by selecting **DynamicLink > Add Netlist File Include** .



NetlistInclude
NetlistInclude1
IncludePath= /hfs/d1/local/users/crscott
IncludeFiles[1]=function.inc
UsePreprocessor=yes

The *IncludePath* parameter should contain the path of the ASCII file and the *IncludeFiles* parameter should contain the file name. When this component is netlisted by ADS, it generates a **#include** statement that is later replaced by the contents of the ASCII file. For more information on file inclusion, refer to *File Inclusion* (cktsim).
The Netlist Include component can thus be used to append a file containing multiple models or even the entire set of models. It can also be used to select among various files containing different sets of process parameters corresponding to different corner cases. For more information, see *NetlistInclude (Netlist File Include Component)* (ccsim).
In a practical example, *typical.inc* could contain the process parameter values (sheet resistance, area capacitance, etc.) for the typical case, while *maximum.inc* would have definitions corresponding to the maximum case. The Netlist Include component can then be used to select which corner case to simulate by pointing to either *typical.inc* or *maximum.inc* .
If you are using RF Design Environment, you can include files into the netlist by selecting **Setup > Model Libraries** in the Affirma Analog Circuit Design Environment window to access the Model Library Setup form. For more information on Model Library Setup, refer to your Cadence documentation.

3. Include the ASCII file with the expression definitions directly in the model file.

```
model npn BJT NPN=yes Is=4.598E-16 Bf=175 Nf=0.9904 Vaf=22 Ikf=0.8 \
...
Xcjc=0.43 Tf=1e-11 Xtf=50 Vtf=test\(AAA\) Itf=0.32 Ptf=32 \
Tr=1E-09 Fc=0.6
#include "/users/home/functions.inc"
```

Note
If an expression is used to define a model parameter, the argument cannot be another model parameter or an instance parameter. If the model needs to use the value of an instance parameter in the calculation of a model parameter, this requires creating a subcircuit that incorporates the model, as in the following example:

```
define npn1 \( c b e \)
parameters AAA=1 area=1 region=1 noise=1
model NPN BJT NPN=yes Is=4.598E-16 Bf=175 Nf=0.9904 Vaf=22 Ikf=0.8 \
Ise=1.548E-14 Ne=1.703 Br=76.1 Nr=0.9952 Var=2.1 \
Ikr=0.02059 Isc=3.395E-16 Nc=1.13 Rb=8 Irb=8E-05 \
Rbm=3 Re=0.45 Rc=6 Xtb=0 Eg=1.11 Xti=3 Cje=8.7E-13 \
Vje=0.905 Mje=0.389 Cjc=3.6E-13 Vjc=0.4907 Mjc=0.2198 \
Xcjc=0.43 Tf=1e-11 Xtf=50 Vtf=test\(AAA\) Itf=0.32 Ptf=32 \
Tr=1E-09 Fc=0.6
NPN:qin c b e 0 Area=area Region=region Noise=noise
end npn1
```

Migrating simInfo Definitions

The ADS simInfo definition for RF Design Environment (RFDE) has been changed slightly from RFIC Dynamic Link to more closely approximate the Spectre simInfo definition. The change has the benefit of enabling some functionality in OASIS direct which would otherwise remain broken, due to sub-classing of OASIS direct netlisting methods.

Important
It is recommended that you update your simInfo by following the steps below if backward compatibility is not required. Performing these changes will prevent your library from being used in ADS 2002A or previous versions of ADS.

In order to migrate from an old RFIC Dynamic Link definition to the new RFDE definition, the following steps should be performed:

1. Update componentName so it no longer uses *expr(iPar('model))* . The model parameter will be accessed through nlGetComponentName. If a parameter other than model was specified in iPar, it is necessary to create a propMapping that will map the name of the parameter used to model.
2. Change the netlist procedure IdfDevPrim or IdfCompPrim to ADSsimCompPrim. Both IdfDevPrim and IdfCompPrim call ADSsimCompPrim so netlisting can speed up by calling ADSsimCompPrim directly.
3. If you are migrating from a Cadence 4.4.3 library, macroArguments must be added to the instParameters field.
4. The namePrefix field is no longer used, and can be eliminated.
5. The trise parameter has been added to all ADS simulator primitives so that you can now use trise/dtemp instead of Temp.

Using the Migration Tool

In order to facilitate the process of migrating an existing Cadence library that contains Dynamic Link simulator definitions, a migration tool has been developed. This migration tool enables the specification of a library, and it will update any ads simInfo definitions it finds within that library. The tool will report an error message in only one case, which is if expr(iPar('<param>)) notation has been used for a componentName, and the parameter name specified was not model. In this case, it will still clear the componentName so it will operate off of model.

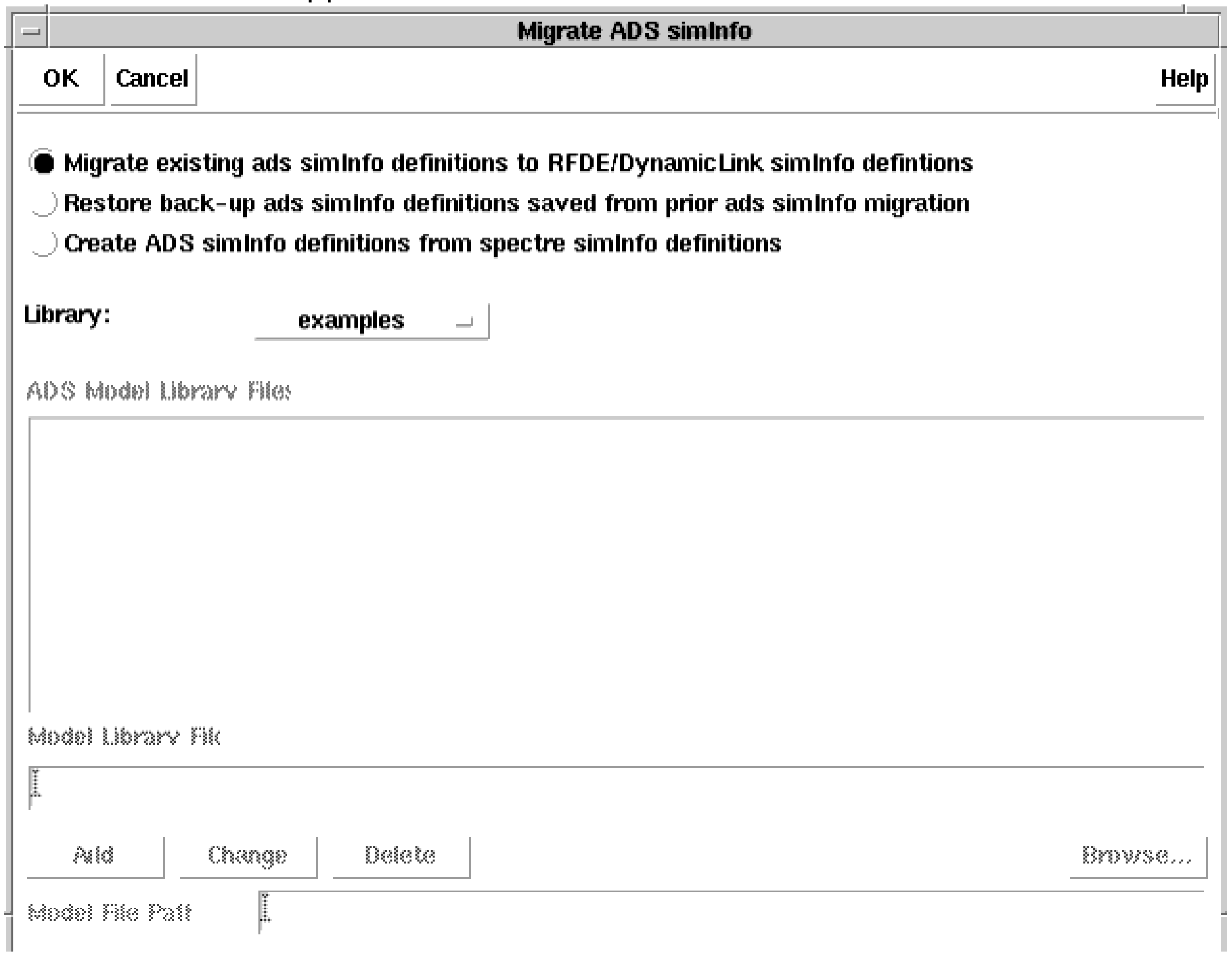
If simInfo fields were added to the default list of ads simInfo parameters, the migration tool will clear them. No warning or error is generated to indicate that a field is being deleted.

In order to enable the ADS simInfo migration tool so it appears as a menu option, it is necessary to set the environment variable RFDE_MIGRATE to *yes* . When this environment variable is set, the **Migrate simInfo** menu item appears under the **Tools > ADS Dynamic Link** menu item on the CIW.

Migrating an Existing ads simInfo Definition

To perform the migration:

1. Choose **Tools > ADS Dynamic Link > Migrate simInfo** on the CIW. The Migrate ADS simInfo form appears.



Migrate ADS simInfo

OK | Cancel | Help

[● Migrate existing ads simInfo definitions to RFDE/DynamicLink simInfo defintions]
Restore back-up ads simInfo definitions saved from prior ads simInfo migration
Create ADS simInfo definitions from spectre simInfo definitions

Library: examples

ADS Model Library Files

Model Library File

Add | Change | Delete | Browse...

Model File Path

   Optionally, you can call the function **rfde_MigrateSimInfoDialog()** from the CIW.
2. Enable the *Migrate existing ads simInfo definitions to RFDE/Dynamic Link simInfo definitions* , and select the appropriate library. Note that analogLib and basic do not appear in the library list since these libraries are delivered with the product, and have already been updated.

Restoring an ads simInfo Definition

After simInfo definitions are migrated, the old definition is stored in a supplemental simInfo section called *oldads* . This enables you to restore a definition if it is decided that the new definition was not what you were really looking for.
To restore a definition:

1. Enable the *Restore back-up ads simInfo definitions saved from prior ads simInfo migration* on the migration form.
2. Click **OK** in the Migrate ADS simInfo form.

Migrating a Spectre Definition to an ADS Definition

Improvements to the ads simInfo definition provide a much closer match to the existing spectre simInfo. While the migration tool is not a complete translator, it does enable you to more reliably automate the process of translating a spectre simInfo definition into an equivalent ads simInfo definition.

Migration is performed on all components in the library that have a spectre stop view. The spectre view is first copied to an ads view in the migration. Afterwards, the simInfo section of the CDF will be set up based on the spectre simInfo section of the CDF.

There are a few things that will be changed in the simInfo of a *spectre* view to get an equivalent *ads* view. Note that this is not always possible, and that the spectre to ads migration tool is provided as an ease of use tool that will get things started, not as a 100% guaranteed translator. The migration tool is primarily provided for translating process design kits (PDKs), and expects that the majority of the components encountered will have an ads simulator primitive that is identical to the spectre primitive, because a subcircuit definition has been created for the component.

To convert a Spectre simInfo into an ADS simInfo:

1. Copy the *spectre* view to an *ads* view if a spectre view exists.
2. Set up the netlist procedure. For compatibility with ADS Dynamic Link 2002, set the netlist procedure to:
    - *IdfCompPrim* if component name is set to a value other than subcircuit.
    - *IdfCompPrim* if model has a default value, and component name is left blank.
    - *IdfCompPrim* if there are no hierarchical views (i.e. extracted or schematic type views).
    - *IdfSubcktCall* if none of the other cases are true.
      This assumes that a special spectre function was not used. Generally speaking, it is safe to use IdfCompPrim for any component.
      If ADS Dynamic Link 2002 compatibility is not an issue, set the netlist procedure to ADSsimCompPrim or ADSsimSubcktCall.
3. Set up the *componentName* field. If a hierarchical subcircuit, netlist included subcircuit, or model parameter is used to determine the component name, use the spectre value directly. Simulator primitives that use models will still use the value of the parameter model, so the spectre value can still be used. If the component name specifies a primitive that does not use a model, the appropriate ADS primitive must be substituted for the spectre primitive name.
   Spectre & ADS Equivalent Primitives shows the ADS primitive equivalent for spectre primitives that do not use models.

Spectre Primitive	**ADS Primitive**
capacitor	C
delay	VCVS
inductor	L
iprobe	Short
mutual_inductor	Mutual
relay	SwitchV
resistor	R
transformer	tf
vccs	VCCS
vcvs	VCVS

The primitives in *Primitives Mapped to Agilent analogLib* have been mapped in the Agilent Technologies version of analogLib. These components are extremely difficult to map because they require special processing in the form of custom netlisting functions. If you have used any of these primitives, it is recommended that you determine an appropriate mapping based on what is provided in analogLib, or simply use the analogLib definition.

Spectre Component	**analogLib component to refer to**
Cccs	cccs
Ccvs	ccvs
Isource	isource
Port	port
Vsource	vource

Spectre Primitives with no Direct ADS Equivalent shows Spectre primitives that have no direct ADS equivalent (i.e. you must construct an equivalent circuit to represent the component, either because the ADS equivalent primitive has dissimilar parameters, or because an SDD must be created to represent the primitive).

Spectre Component	**Can not be used because...**
a2d	No equivalent
cktrom	No equivalent
d2a	No equivalent
fourier	No equivalent
intcap	C Model in ADS does not have equivalent parameters
msline	ADS mlin requires msub, too disimilar to use
node	No equivalent
nport	No equivalent, custom S-parameter circuit must be written
paramtest	Equivalent functionality can be achieved using if statements, but cannot be done as a primitive
pcccs	Requires mapping to an SDD
pccvs	Requires mapping to an SDD
phy_res	No equivalent
pvccs	Requires mapping to an SDD
pvcvs	Requires mapping to an SDD
quantity	Option statements could be used, but the options are not equivalent
rdiff	R Model in ADS does not have equivalent parameters
scccs	No equivalent
sccvs	No equivalent
svccs	No equivalent
svcvs	No equivalent
switch	SwitchV only allows 2 settings, is not quite equivalent
tline	ADS tlin4 requires different parameters
winding	No equivalent
zcccs	No equivalent
zccvs	No equivalent
zvccs	No equivalent
zvcvs	No equivalent

4. Set up the terminal order. If a subcircuit is being used, use the spectre terminal order. Generally speaking, the ADS terminal order is identical to the spectre terminal for primitives. Consult *Terminal Order* to see what the equivalent terminal order for an ADS component is versus a supported spectre primitive.

Spectre Component	Spectre Term Order	ADS Term Order
b3soipd	d g s b	d g s b
bjt	c b e [s]	c b e [s]
bsim1	d g s b	d g s b
bsim2	d g s b	d g s b
bsim3	d g s b	d g s b
bsim3v3	d g s b	d g s b
bsim4	d g s b	d g s b
btasoi	d g s b	d g s b
capacitor	pos neg	pos neg
delay	p n ps ns	p n ps ns
diode	pos neg	pos neg
gaas	d g s	d g s
hvmos	d g s b	d g s b
inductor	pos neg	pos neg
iprobe	pos neg	pos neg
jfet	d g s [b]	d g s
mos0	d g s b	d g s b
mos1	d g s b	d g s b
mos2	d g s b	d g s b
mos3	d g s b	d g s b
mos15	d g s b	d g s b
mutual_inductor	<no terminals>	<no terminals>
relay	p n ps ns	p n ps ns
resistor	pos neg	pos neg
tom2	d g s	d g s
transformer	t1 b1 t2 b2	t1 b1 t2 b2
vbic	c b e [s] [dt] [tl]	c b e [s] [dt]
vccs	p n ps ns	p n ps ns
vcvs	p n ps ns	p n ps ns

5. Set up the terminal mapping. The terminal mapping for ADS is constructed in a similar manner to the spectre terminal mapping. However, in ADS, pin annotations are always output using a numeric name (spectre does custom name some pin outputs, like on the BSIM where the spectre names are d g s b). ADS pin numbers always start with 1, and increment from there. To make the ADS mapping, take the number of the terminal name in the list, then add ": P" to that number.
   Example: termOrder='(D G S B)
   termMapping='(nil D ": P1" G ": P2" S ": P3" B ": P4")
   All two terminal primitives in ADS will only output a positive direction current. For these components, it is necessary to apply the minus operator to the negative terminal. In Spectre, these are formatted as (FUNCTION minus(ROOT "PLUS")). ADS uses a slightly different syntax. To map a terminal to be the negative of a positive terminal, place the keyword "minus." between the colon and the positive terminal name.
   Example: termMapping='(nil PLUS ": P1" MINUS ":minus.P1")
6. Set up the instance parameters. The instParameters field for subcircuits will match

the spectre instParameters field, with one exception. It is necessary to map the multiplicity parameter *m* to *_M* . If a simulator primitive is used directly, it is necessary to set up the instParameters field with the proper parameter names for ADS. Unlike spectre, ADS uses mixed case names or upper case names. This means that the spectre primitive parameter names will never match the spectre primitive names, which are always lower case. The parameter names are also not always identical.
Consult *Instance Parameter Mapping* to see what the equivalent parameter names are for instance primitives. Note that parameters that have no applicable mapping to ADS are not listed.

Spectre Component	**Spectre Instance parameters**	**ADS Instance Parameters**
b3soipd	w l as ad ps pd nrd nrs region m	W L As Ad Ps Pd Nrd Nrs Region _M
bjt	area region trise m	Area Region Trise _M
bsim1	w l as ad ps pd nrd nrs region m	W L As Ad Ps Pd Nrd Nrs Region _M
bsim2	w l as ad ps pd nrd nrs region m	W L As Ad Ps Pd Nrd Nrs Region _M
bsim3	w l as ad ps pd nrd nrs region m	W L As Ad Ps Pd Nrd Nrs Region _M
bsim3v3	w l as ad ps pd nrd nrs region m	W L As Ad Ps Pd Nrd Nrs Region _M
bsim4	w l as ad ps pd nrd nrs region m	W L As Ad Ps Pd Nrd Nrs Region _M
btasoi	w l as ad ps pd nrd nrs region m	W L As Ad Ps Pd Nrd Nrs Region _M
capacitor	c tc1 tc2 trise tnom w l m	C TC1 TC2 Trise Tnom Width Length _M
delay	td gain m	T G _M
diode	area perim region trise m	Area Periph Region Trise _M
gaas	area region m	Area Region _M
hvmos	w l as ad ps pd nrd nrs region m	W L As Ad Ps Pd Nrd Nrs _M
inductor	l r tc1 tc2 trise tnom m	L R TC1 TC2 Trise Tnom _M
iprobe	No Parameters	
jfet	area region m	Area Region _M
mos0	w l m	W L _M
mos1	w l as ad ps pd nrd nrs region m	W L As Ad Ps Pd Nrd Nrs Region _M
mos2	w l as ad ps pd nrd nrs region m	W L As Ad Ps Pd Nrd Nrs Region _M
mos3	w l as ad ps pd nrd nrs region m	W L As Ad Ps Pd Nrd Nrs Region _M
mos15	w l as ad ps pd nrd nrs region m	W L As Ad Ps Pd Nrd Nrs Region _M
mutual_inductor	coupling ind1 ind2	K Inductor1 Inductor2
relay	vt1 vt2 ropen rclosed m	V1 V2 R1 R2 V2_M
resistor	r l w tc1 tc2 trise isnoisy m	R Length Width TC1 TC2 Trise Noise _M
tom2	area region m	Area Region _M
transformer	n1 n2 m	
vbic	area region trise m	Area Region Trise _M
vccs	gm m	G _M
vcvs	gain m	G _M

7. Set up property mapping. Consult the table provided in section (6). The property mapping is a disembodied list, with the output instance name first, followed by the parameter name that is mapped to that instance parameter. If spectre uses a property mapping, make sure to use that value, not the spectre instance parameter name.
Example: propMapping='(nil W w L l as As ad Ad Ps ps Pd pd Nrd nrd Nrs nrs Region

region _M m)

8. Set up type mapping. The type mapping enables you to specify a function that will be called to process the value of a parameter prior to netlisting. This may be necessary for some parameters that netlist differently for spectre versus ADS (e.g. Region in ADS is an integer, but is a string literal in spectre). Additionally, in Cadence 4.4.6, there is a simInfo property, *stringParameters* . All of the parameters specified in stringParameters should be set up to output as type "string".
   In ADS, a type mapping function is created by prefixing *rfdeNetlistTypemap_* to the name of the mapping you wish to use. Thus, a function *rfdeNetlistTypemap_string* would be referred to as "string" in the type mapping field.
   ADS provides three relevant type mappings:
   - **string** - The string type mapping should be used if a parameter value needs to be output with double quotes.
   - **region** - The region type mapping takes the typical cyclic field strings used on the region parameters in analogLib, and maps them to an appropriate integer value.
   - **boolean** - The boolean type mapping will take a values of "t" and output it as "yes", and will otherwise output "no". In ADSsim, "yes" corresponds to integer 1 and "no" corresponds to integer 0.
     Example: typeMapping='(nil Region region)
9. Set up uselib. The *uselib* field specifies that a subcircuit from an ADSsim library needs to be included in the final netlist. The ADSsim libraries are special libraries that are provided in $HPEESOF_DIR/circuit/components. Use *Components Requiring a uselib Statement* to see the components that require a uselib statement.

ADS Component	uselib value
tf	ckt
VCCS	ckt
VCVS	ckt

To perform the migration:

1. Choose **Tools > ADS Dynamic Link > Migrate simInfo** on the CIW. The Migrate ADS simInfo form appears.
2. Select the *Create ADS simInfo definitions from spectre simInfo definitions* option in the Migrate ADS simInfo form.
3. Select the library to migrate. At present, the migration tool does not support mapping of all spectre primitives. Additionally, certain primitives that use models may be misidentified, resulting in inappropriate parameter mappings. As with the old ads to new ads migration, analogLib and basic are not shown in the library list, because those libraries have been set up and delivered with the product.

Adding CDF/SimInfo to a Component Library

The section provides information on modifying the Cadence simInfo (Simulation Information) section in a CDF (Component Description Format) file.

Using cdfDumpAll

The benefit of adding simulator information via cdfDumpAll is that you need not have numerous files containing specific simulation parameters and simInfo. Instead, all of the CDF information is compiled for you in a single ASCII file. This method is probably your best choice if you do not have source files for parameter and simInfo data for each and every simulator that a library currently supports.

Dumping the CDF for an Entire Component Library

To create and modify an ASCII file containing the entire CDF for an existing component library:

1. Enter the following Skill command in the Cadence CIW:
   ```
   cdfDumpAll("libName" "fileName" ?edit t)
   ```
2. In the text editor of your choice (vi, emacs, etc.), for each library cell add the simInfo for the new simulator ads to the CDF file. In some cases, you may also need to add new CDF parameters.
3. Load this file in the CIW using the command:
   ```
   load "fileName"
   ```

This modifies the library database accordingly, assuming you have write permission to the library.

Dumping the CDF for Individual Components

To create and modify an ASCII file containing the CDF for an individual component:

1. Enter the following Skill command in the Cadence CIW:
   ```
   cdfDump("libName" "fileName" ?cellName "cellName" ?edit t)
   ```
2. In the text editor of your choice (vi, emacs, etc.), for each library cell add the simInfo for the new simulator ads to the CDF file. In some cases, you may also need to add new CDF parameters.
3. Load this file in the CIW using the command:
   ```
   load "fileName"
   ```

This modifies the library database accordingly, assuming you have write permission to the library.

Cadence Documentation References

The following references supplement the information in this section. All Cadence documentation is available from the Cadence Documentation Browser (cdsdoc) and/or Openbook.

1. Cadence Component Description Format User Guide
2. Cadence Library Manager User Guide
3. Cadence Analog Design Environment SKILL Language Reference
4. Cadence SKILL Language Reference
5. Cadence SKILL language User Guide

The basic Library

In prior releases, RFIC Dynamic Link required that the basic library *nlpglobals* cell contained the *ads* view. In the past, the software was distributed with a modified version of the basic library with a special nlpglobals view that included an ADS stop view. This is no longer required in the current release or future releases of RFIC Dynamic Link.

Modifying the analogLib Library

RFIC Dynamic Link includes a version of the Cadence analogLib that has been extended to work with the ADS Analog/RF Simulator (ADSsim). This library is located in:

```
$HPEESOF_DIR/cdslibs/5.1.0/analogLib
```

which only works with Cadence IC 5.1.41 (CDBA version). For more information on the Agilent Technologies analogLib components, refer to the analogLib Components documentation.

Note
It is strongly recommended that you use the Cadence analogLib library in place of the Agilent analogLib components.

If you need to extend your own version of analogLib to work with the ADSsim, this section may be useful.

Note
Agilent Technologies has performed extensive updates to the modified version of analogLib. If you create your own version of analogLib, Agilent Technologies cannot be responsible for problems that may occur as a result of the modified library.

To modify your version of analogLib:

1. Copy your current version of analogLib to a new working version of analogLib.
2. Start Cadence while pointing to your new working version of analogLib.
3. Load the skill file.

   ```
   $HPEESOF_DIR/cdslibs/skill/analogLib_siminfo.il
   ```

   This will automatically add the necessary simInfo definitions and create the appropriate stop view for your new working version of analogLib.

Using the procedure above will help to ensure that you do not loose any modifications you have made to your existing analogLib.