Agilent Technologies

Advanced Design System 2002

Series IV Design Translation

February 2002

Notice

The information contained in this document is subject to change without notice.

Agilent Technologies makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Agilent Technologies shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Warranty

A copy of the specific warranty terms that apply to this software product is available upon request from your Agilent Technologies representative.

Restricted Rights Legend

Use, duplication or disclosure by the U. S. Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 for DoD agencies, and subparagraphs (c) (1) and (c) (2) of the Commercial Computer Software Restricted Rights clause at FAR 52.227-19 for other agencies.

Agilent Technologies
395 Page Mill Road
Palo Alto, CA 94304 U.S.A.

Contents

Chapter 1: Series IV Migration Overview

ADS 2001 contains a utility for migrating from Series IV to ADS. Migrating from Series IV to ADS is a process that involves moving your entire Series IV design process into ADS. This includes migrating custom libraries, models, and AEL macro scripts, as well as translating design files and projects. This manual describes the migration process and the design file translation utility. Once you have successfully migrated your design process, you can take advantage of the full power of ADS with your existing designs.

The translation process is simple in theory; the Series IV components are replaced with their ADS equivalents. While it is logical to assume that ADS will yield the same simulation results as Series IV after translation, there are many reasons for minor differences between the results. Some components (particularly nonlinear ones) have improved models in ADS. The ADS simulators also have been improved and have new algorithms. Finally, there are differences in the way the tools define simulations and parameters. The translation utility may not be able to resolve these differences. To ensure that your ADS simulation results match those in Series IV, you should understand the basic differences between the two tools.

The next section highlights the main differences between Series IV and ADS. Please read it carefully before beginning any design translation. Detailed information about specific model differences and items that are not translated can be found in the following appendices:

- Appendix A, Series IV Items Not Translated
- Appendix B, Nonlinear Model and Component Changes from Series IV to ADS

The information and procedures documented here assume you are familiar with both Series IV and ADS. If you have never used ADS before, review the online *Quick Tour* to familiarize yourself with the basics of ADS.

Changes Between Series IV and ADS

This section describes some of the changes that you should be aware of between Series IV and ADS. Some areas may require manual changes after the translation is complete.

Table 1-1. Series IV to ADS Changes

Defining Schematic and Parameter Units
Dimensional units and scale factors for component parameters were defined in Series IV through the Defaults window or in a design's UNITS item. ADS has no direct equivalent to the Defaults window or UNITS item. Instead, units and scale factors are defined individually for each parameter. The default ADS units are defined through Options > Preferences in the Schematic (or Layout) window. This difference can cause translation problems if hierarchical designs contain different unit definitions than the Defaults window. During translation of a project, the units from the Defaults window are used; all other units are ignored.
Test Benches and Test Labs
In Series IV, simulations and many measurements were defined in test benches and test labs. There are no direct equivalents to these in ADS; the translator creates hierarchical schematics where an instance of the schematic is placed in the simulation design. Note that ADS simulation controls may not offer exact duplication of functions. In particular, measurements and measurement equations will not translate in a functional manner.
Nonlinear Model Data Items
For differences between the Series IV version and the ADS version, refer to Appendix B, Nonlinear Model and Component Changes from Series IV to ADS.
Conductivity Values
In Series IV, conductivity (RHO) was specified relative to gold. Gold has a conductivity of 4.1E7 Siemens/meter and in ADS conductivity is expressed in Siemens/meter. Note that conductivity values are converted during translation. Note the following values used in ADS: 5.80E7, Copper 4.10E7, Gold 6.20E7, Silver 3.80E7, Aluminum
Output Equation Measurements
The Series IV OUT_EQN item is obsolete, but it is translated to a MeasEqn item.
OmniSys Designs
ADS (beginning with version 1.0) provides an alternate and much more powerful alternative to the Series IV OmniSys simulator. There are major differences between the two tools that prevent automatic translation of these designs Only certain passive OmniSys models that were available for use in Series IV circuit designs can be translated.

Table 1-1. Series IV to ADS Changes

Series IV Data Items

The items listed below were placed in the Defaults window in Series IV. This concept does not exist in ADS and these items are not translated as individual components. The table below identifies how this information is brought over in ADS.

Data Item	Notes
NPAR	No ADS equivalent
MCOVER	Translated as the Hu parameter of the MSUB component
MWALL	Translated MLEF, MLIN, MLOC, MLSC
MDIFLIST	Translated as the iVar*n*, iVal*n* parameters of S2PMDIF, where *n* = 1 to 10.
PERM	Translated as the Mur parameter of substrate components (such as MSUB)
PWLSRC	Parameter information included in ItPWL or VtPWL component
RREF	Translated as individual parameter for individual components (e.g., Z1, Z2 in Amplifier)
SIGMA	Translated as an individual parameter (Sigma) for individual components
TAND	Translated as the TanD parameter of substrate components (such as MSUB)
TEMP	Translated as the Temp parameter of components that reference a substrate component (such as MACLIN)

Differences Between Harmonic Balance Simulators

While the harmonic balance simulators used by Series IV and ADS are conceptually the same, there are implementation differences that can potentially result in simulation differences between the two. This applies particularly to the FFT sampling algorithms in the nonlinear simulators.

To ensure the best possible agreement between Series IV and ADS Harmonic Balance analyses, the full harmonic content of the circuit under test must be characterized. If this is not ensured, Series IV and ADS harmonic balance results will have significant, little, or no difference at all; this strictly depends upon the amount of nonlinearity in the circuit under test.

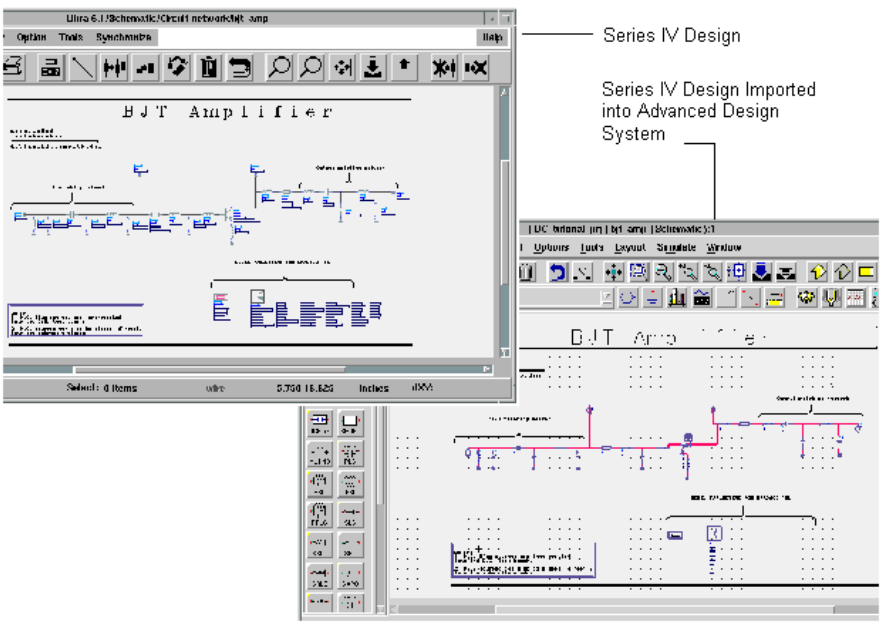
Table 1-1. Series IV to ADS Changes

S-parameter Data Interpolation
You must ensure that S-parameter (or any other frequency-dependent) data files contain all frequencies that are encountered in the simulation, you must keep in mind that a Harmonic Balance simulation will run the analysis at DC and at the highest harmonic specified. Consequently, S-parameter files must have these frequencies specified. When DC, or higher-frequency data, is not present, the simulator will extrapolate. Extrapolated results between Series IV and ADS can potentially be different because of implementation differences.
Phase Noise Differences
Phase noise simulation technology is very abstract and its development is presented with strong technical challenges. Consequently, since the time it was first introduced into Agilent-EEsof products, it has undergone a maturation process. In ADS, to address this, and, to instill a sense of confidence for the user, ADS offers two separate phase noise methods, pnmx and pnfm (Mixing analysis and FM sensitivity analysis, respectively). Refer to the ADS Circuit Simulation manual for details. Series IV, like ADS, also uses the pnmx and pnfm phase noise algorithms. However, there are limitations with the Series IV implementation. Below are those that can significantly contribute to ADS/Series IV phase noise discrepancies. -The pnfm models in Series IV are deficient, lacking several important terms. -The pnmx model is implemented using only half of the correct number of small signal sidebands. One of several problems incurred with this is that it is more susceptible to the flattening problem at low offset frequencies. -Series IV mistakenly adds the pnfm and pnmx results together, instead of showing them separately.
Nonlinear Vendor Library Component Simulation Temperature
In Series IV the default temperature used was 27 degrees C. In ADS the default temperature used is 25 degrees C. If Series IV designs containing nonlinear vendor library components relied on the default temperature, your simulation results will vary. The workaround in ADS is to set the desired temperature via the Options Controller (available in most simulation controller libraries).

Chapter 2: Importing and Simulating

This chapter outlines the basic steps for successfully using the Series IV design translation utility. Details about the translator and its configuration are described in Appendix C, Translator Customization. Known defects and areas that may require manual changes at some point in the translation process are described in Appendix E, Known Issues.

Note If the design or project you want to translate references designs that reside in another location, such as a library, then you must translate those designs first. For details, refer to Chapter 3, Translating Design Libraries. The same is true for designs/projects that reference Series IV user-defined models. For details refer to, Chapter 4, Updating User-Defined Models.



Importing Designs

The translator allows you to translate designs individually or collectively. However when you translating designs individually, take the following into consideration:

- If you select an individual design to translate, and that design is hierarchical, the subnetwork designs are also translated. Note that if you translate once, go back to Series IV and make any changes to either the top-level or subnetwork designs and translate again (into the same destination project). Only the top-level (or specified) design is re-translated.
- If you translate designs individually, and any of those designs is referenced in multiple test benches, it may cause a translation error when the test benches are translated.
- If different designs in the project are based on different defaults designs, you should import those designs individually rather than as part of a project, because you can only associate one defaults design with a project on import. Importing designs individually enables you to associate unique defaults designs with each.
- If a Units Item exists in the Schematic window for an individual design, that Units Item is used; otherwise, the Units Item from the selected *Defaults* design is used.

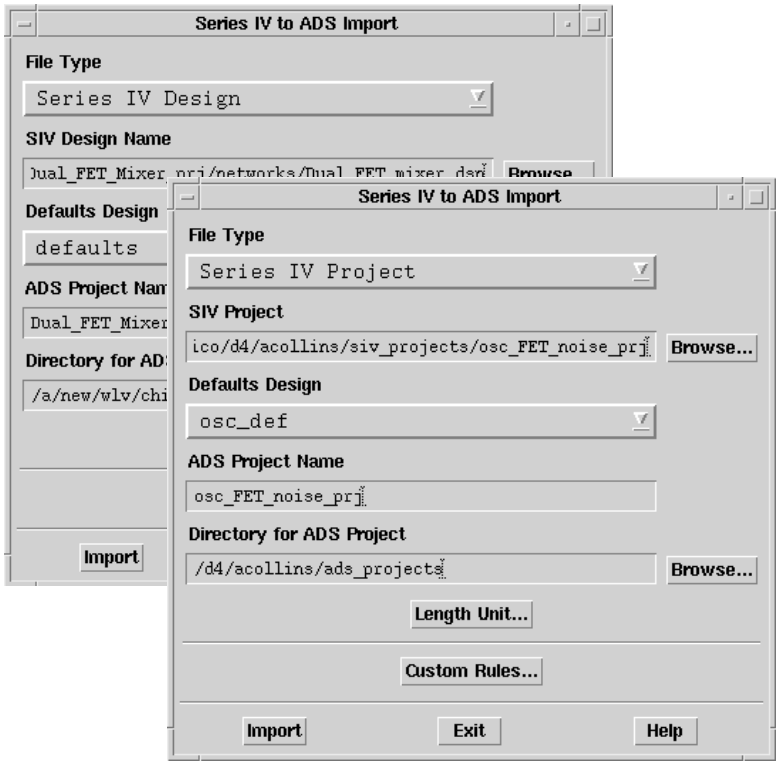
The following considerations apply to translating both individual designs and projects:

- Spaces are not allowed in project paths or project names.
- Series IV design names consisting of or beginning with the word *untitled* (such as *untitled1.dsn*) are not allowed. If your project contains any designs named this way, an error message will be displayed.
- Names of built-in ADS components are not allowed as design names. If duplicate names are encountered, your designs will be renamed by the translator (an R_ will be added to the front of any such design names).

To import a Series IV design or project into an ADS project:

1. Start the translator:
   - UNIX—At the prompt type **siv2ads**
   - PC—From the *Start* menu, choose **Programs** > **Advanced Design System 2001** > **ADS Tools** > **Series IV Import**.

2. To begin the import process, select **Series IV Design** or **Series IV Project** from the File Type drop-down list.



3. Identify the SIV Design Name or the Series IV Project by entering the path and filename or project name, respectively, or use the Browser to select it.
4. If more than one defaults design exists in the project, select the one you want to associate with the individual design, or if importing a project, all designs in the project.
5. In the ADS Project Name field, accept the proposed ADS project name or specify a different one.
6. In the Directory for ADS Project field, type the complete path, or use the Browser to set the path.
7. Click **Length Units** and select the appropriate setting. Note: This setting establishes a default for components placed in the new project, beyond

translation. This unit setting serves as a default for all designs in the project and is both:

- The unit of measure for parameters with physical length (in both Schematic and Layout windows)
- The design unit (grid display and cursor snapping) in the Layout window

---

Note The unit used for grid and snap features in the drawing area of the Schematic window is inches, and cannot be changed. When you see choices in various dialog boxes for *screen pixels* or *schem units*, the *schem units* are inches.

Click **OK** to establish Length Units.

8. If the design(s) you are translating reference any user-defined models for which you have created one or more customized rules files, click **Custom Rules**. Click **Build Database**. When it is done building, select the option **Use Custom Database**, and click **OK**. For details on creating and using customized rules files, refer to "Using a Customized Rules Files" on page C-8.
9. Click **Import** to begin the import process.

   The Status window displays feedback during the import process. This information is also written to a file, *me_prj.log*. Detailed information about the translation can be found in the file *me_err.log*.

Note These files are created in the ADS project directory and are overwritten with each subsequent individual design translation into this project.

Simulating Imported Designs

Advanced Design System simulations are performed within the Schematic window. All data produced by a given simulation is stored collectively as a *data set*. Every data set has a name associated with it. You can assign a name prior to simulating or you can accept the default data set name. The default name is the name of the current design, with the extension *.ds* automatically appended.

Note If you accept the default data set name and perform multiple simulations, the data set will be overwritten each time. To collect separate data sets for each simulation, specify a unique name (for the data set you are about to create) prior to each simulation.

To specify a data set name prior to simulating:

1. Choose **Simulate** > **Setup**.
2. Supply a name in the *Dataset* field. (Click Browse to view existing dataset names in the current project.) Note that data sets are stored in the */data* subdirectory of a project. This means that different designs in the same project directory should all generate data sets with unique names.
3. Optionally, select a different *Remote Simulation Host* for the simulation, if you are running the simulation on a remote machine.

   For details on setting up your remote and local machines for remote processing, refer to the appropriate appendices:

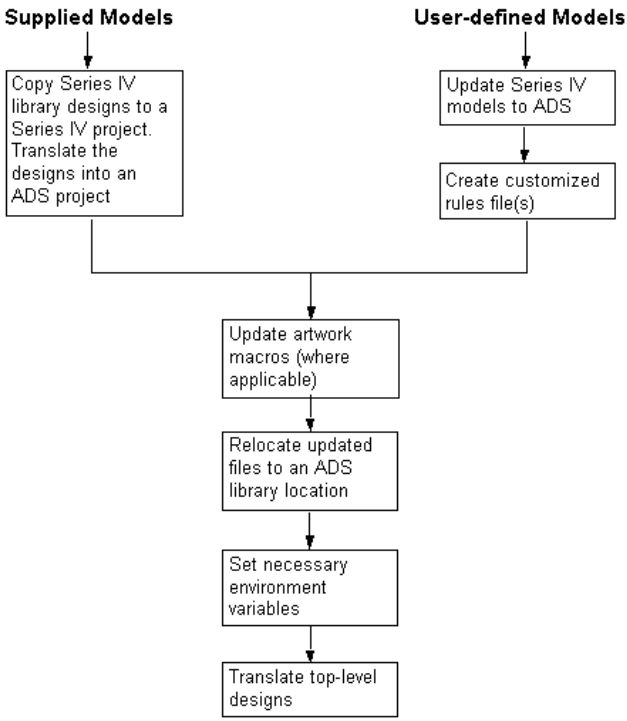
   - Appendix E, Using Remote Simulation in the *Installation on PC Systems* manual
   - Appendix D, Using Remote Simulation in the *Installation on UNIX Systems* manual.
4. Click **Simulate**.

   For information on simulating a design, refer to “Basic Circuit Simulation Procedure” in the Circuit Simulation documentation.
5. Display the simulation results.

   For information on displaying simulation data, refer to “Displaying Simulation Results” in the Data Display documentation.

Chapter 3: Translating Design Libraries

Translating design libraries consists of several steps. If any of the designs are user-defined (as opposed to designs built from supplied models), additional steps are required. For details on that process, refer to Chapter 4, Updating User-Defined Models.



Copying Designs to a Series IV Project

To translate several designs at once, they need to be in a Series IV project, if you have a collection of designs to translate that are not currently part of a project directory (for example, in a directory outside a project that serves as a *library*) you must copy them to a project first. If you plan on translating designs individually, you can leave them in their current location.

Translating Subnetwork Design Files

Launch the translator and select the project (or design) you want to translate, as described in "Importing Designs" on page 2-2.

Note Ignore the *Custom Rules* option for this step. You will select a rules file later when you translate the top-level designs, if they reference user-defined models as subnetworks.

If applicable, update any artwork macros at this time, as described in the section, "Artwork Macros" on page E-1.

Once the subnetwork design (*.dsn*) files are translated, go to the next step, which varies depending on the design type:

- Designs built from supplied models—move them to their new ADS library location, as described in the next section.
- Designs built from user-defined models—update models as described in Chapter 4, Updating User-Defined Models, and return to this point in the process.

Relocating Designs for Site-Wide Use

Once all designs involved have been translated, you can move the designs into ADS library locations as follows:

Move the design files (*.dsn*) to

$HPEESOF_DIR/custom/circuit/symbols

Move the AEL files (*.ael*) to

$HPEESOF_DIR/custom/circuit/ael

Set the variable SITE_AEL to include the search path for the *.ael* files (the */symbols* directory will be searched automatically for *.dsn* files). Example:

```
SITE_AEL = $HPEESOF_DIR/custom/circuit/ael/
```

Set this variable in the file *$HPEESOF_DIR/custom/config/de_sim.cfg*.

For additional information on the settings used in Series IV vs. ADS, as well as settings for a user-level library, see Figure 3-1.

Translating the Top-Level Designs

Launch the translator and select the project (or design) you want to translate, as described in "Importing Designs" on page 2-2.

If your library designs contain user-defined models, you must also select the custom rules file that maps your Series IV user-defined models to ADS user-defined models. For details, refer to Appendix C, Translator Customization.

Figure 3-1 shows typical/recommended library locations and the environment variables that must be modified accordingly. The figure shows settings for both user and site-wide libraries.

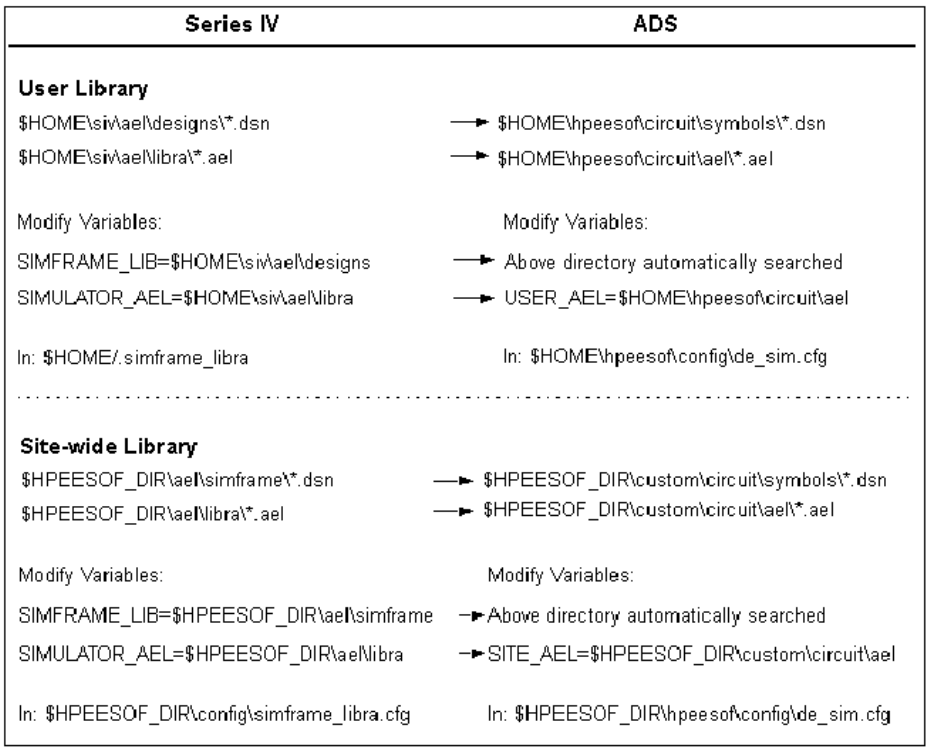


Figure 3-1. Library Settings in Series IV versus ADS

Chapter 4: Updating User-Defined Models

A tool was added to ADS 1.0 to facilitate the creation of user-defined models. However, updating Series IV models for use with that tool requires steps that are not documented here. This process will be documented at a later date and published on the web. The procedure described here uses the translator to update your AEL, but then relies on compiling and linking from the command line (as you did in Series IV).

The example provided here uses actual files from the *$EESOF_DIR/lib/libra/senior* directory to describe the steps required to update your user-defined models from Series IV to ADS; substitute your filenames for the files referred to in the example that are taken from the aforementioned directory. The basic process consists of the following steps:

- Create a custom rules file mapping the Series IV design names to ADS design names
- Translate the project containing the symbol (*.dsn)* and *.ael* files to a new ADS project
- Modify the ADS *user.mak* file and compile

Figure 4-1 depicts the steps involved in this process, using the aforementioned files.

Note If your C-code contains calls to built-in models, additional steps are required. This process will be documented at a later date and published on the web.

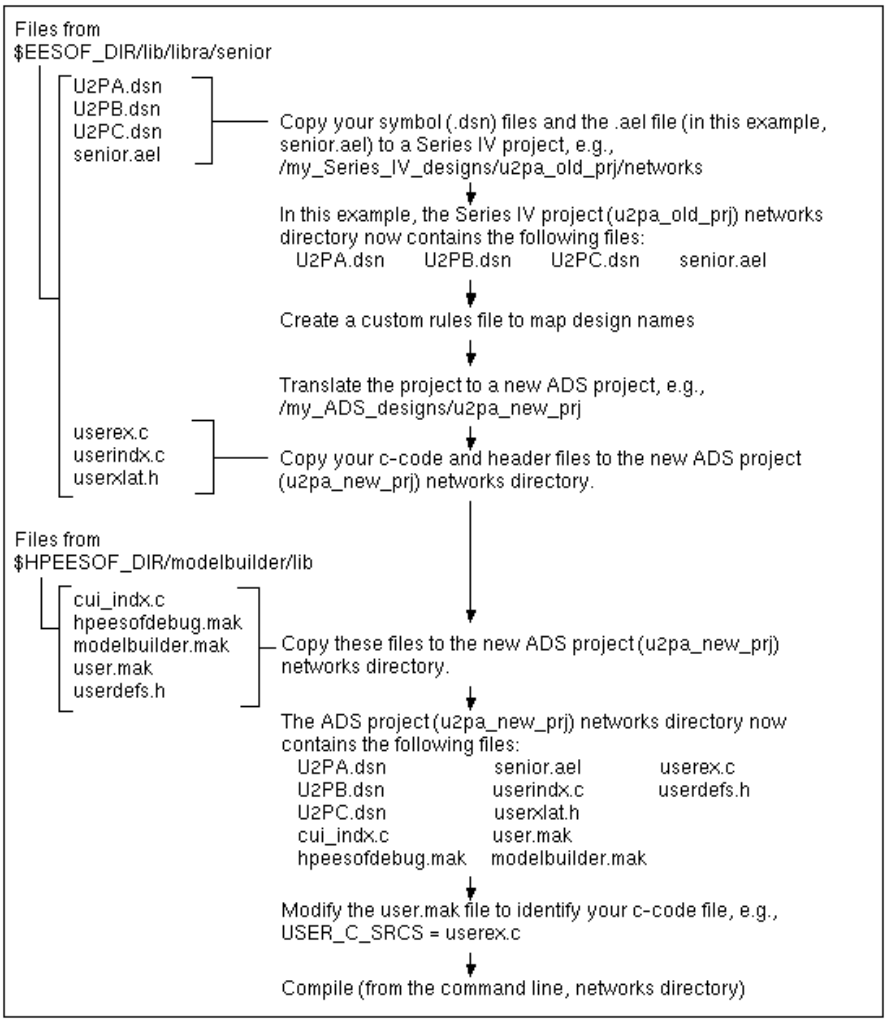


Figure 4-1. Example updating Series IV factory-supplied user-defined models to ADS

To update Series IV models to ADS, do the following:

Note This procedure is written based on UNIX syntax; for PC differences, refer to the ADS *User-defined Models* manual.

1. If the symbol (*.dsn)* files are not in a Series IV project directory, copy them to a Series IV project *networks* directory (from which you will translate). For example, */my_Series_IV_designs/u2pa_old_prj*.
2. Create a custom rules file to map the Series IV design names to ADS design names. In this example, the file looks like this:

```
U2PA|U2PA| | | | |FALSE| |
U2PB|U2PB| | | | |FALSE| |
U2PC|U2PC| | | | |FALSE| |
```

   For details on creating a custom rules file, refer to Appendix C, Translator Customization.
3. Launch the translator and specify old and new projects as described in Chapter 2, Importing and Simulating.
4. Click **Custom Rules**. In the dialog box that appears, select the file you created, click **Add**, and click **Build Custom Database**.

Note The building process takes time to complete; a message will be displayed upon completion.

5. When it is done building, select the **Use Custom Database** option and click **OK**. The translation begins.
6. Copy your C-code file, *userindx.c* file (the one that you modified originally to build the *libra.bin* executable), and header file to the *networks* directory of the new ADS project.

Note If you no longer have your *userindx.c* file, you must recreate it (see template in *$HPEESOF_DIR/modelbuilder/lib)*. In this example, we are using *userex.c, userindx.c, userxlat.h*. from the Series IV *senior* directory.

7. Copy the following list of files from *$HPEESOF_DIR/modelbuilder/lib* to the networks directory of the new ADS project:

cui_indx.c

hpeesofdebug.mak

modelbuilder.mak

user.mak

userdefs.h

8. Due to the error in the *userdefs.h* file noted above, you need to open the file in a text editor and make the changes shown below.

   Replace these lines:

```
extern double get_funit (IN void *eeElemInst);
extern double get_runit (IN void *eeElemInst);
extern double get_gunit (IN void *eeElemInst);
extern double get_lunit (IN void *eeElemInst);
extern double get_cunit (IN void *eeElemInst);
extern double get_lenunit (IN void *eeElemInst);
extern double get_tunit (IN void *eeElemInst);
extern double get_angunit (IN void *eeElemInst);
extern double get_curunit (IN void *eeElemInst);
extern double get_volunit (IN void *eeElemInst);
extern double get_watt (IN void *eeElemInst, IN double power);
```

   With these lines:

```
#define get_funit(eeElemInst)  1.0
#define get_runit(eeElemInst)  1.0
#define get_gunit(eeElemInst)  1.0
#define get_lunit(eeElemInst)  1.0
#define get_cunit(eeElemInst)  1.0
#define get_lenunit(eeElemInst)  1.0
#define get_tunit(eeElemInst)  1.0
#define get_angunit(eeElemInst)  1.0
#define get_curunit(eeElemInst)  1.0
#define get_volunit(eeElemInst)  1.0
#define get_watt(eeElemInst, power)  (power)
```

Note In Series IV these declarations returned multipliers to convert parameter data to SI units. ADS uses scale factors on individual parameters, thus these declarations are obsolete. Replacing the lines as shown returns unity. When you translate your *.ael* files, scale factors are added to individual parameters based on the unitCode in the *create_parm* function and the settings of the UNITS item in the *Defaults Design* you select when translating.

9. Using any text editor, modify the *user.mak* file to identify your C-code file. In this example, this means setting the line *USER_C_SRCS* to point to the file *userex.c*, as shown next.

   ```
   USER_C_SRCS = userex.c
   ```

   Save the file.

10. Compile from the command line by typing

    hpeesofmake -f hpeesofdebug.mak <**target**>

    where *target* is one of the following (if no target is specified, the process defaults to the *compile_and_link* target):

compile_only	compile only the active model
link_only	link without compiling anything
compile_and_link	compile the active file and link everything
update_only	compile the active and any out-of-date files
update_and_link	compile the active and any out-of-date files and then link

    This step assumes you have a path set for the C-compiler.

Note The C-code and header files, and the files used for compiling and linking, do not need to reside in the *networks* directory with the symbol (*.dsn*) and *.ael* files (as was done in this example). These files can be in any directory you choose as long as the affected projects have a soft link to the new simulator executable (*hpeesofsim*) created via the compiling process.

Chapter 5: Translation Example

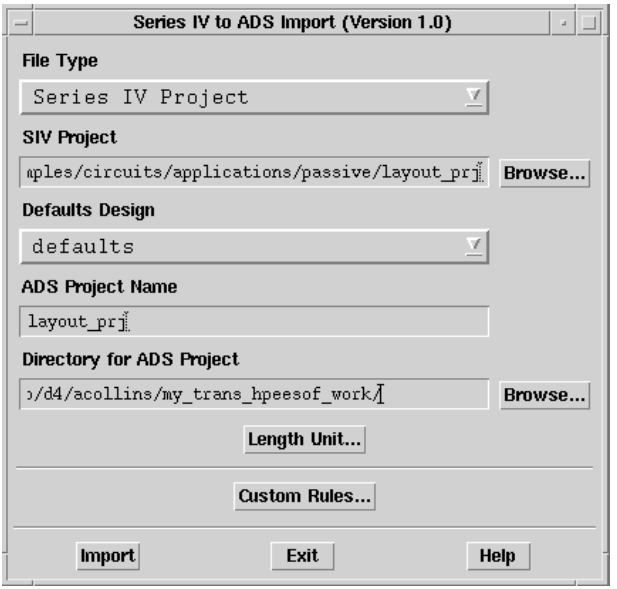
This example shows you how to translate a simple side-coupled filter from Series IV to ADS. The project used is from the following Series IV examples directory:

$EESOF_DIR/examples/circuits/applications/passive/layout_prj

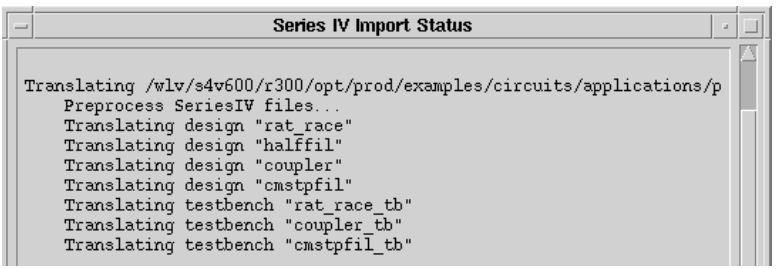
where *$EESOF_DIR* represents your complete installation path.

To translate this project:

1. Start the translator.
   - UNIX—At the prompt type **siv2ads**
   - PC—From the *Start* menu, choose **Programs** > **Advanced Design System 2001** > **ADS Tools** > **Series IV Import**
2. When the Series IV to ADS Import dialog box appears, from the File Type drop-down list, select **Series IV Project**.
3. Use the Browse button to locate the Source Project, **$EESOF_DIR/examples/circuits/applications/passive/layout_prj**.
4. Accept the proposed *Defaults Design*.
5. In the ADS Project Name field, accept the default project name (same as the Series IV name) or enter a new project name.
6. Use the Browse button to adjust the path or type a directory name for the new project.



7. Click **Import**. The Status window appears, reporting the progress.



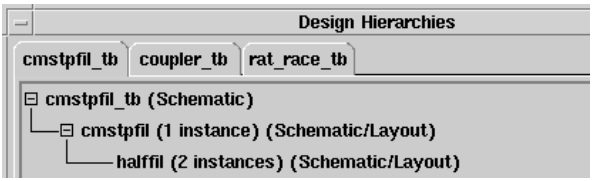
A log file (*me_proj.log*) is created in the new project directory during the translation process. It contains detailed information on component name and parameter changes made to the translated designs. More detailed component information can be found in the *me_err.log* file in the same project directory.

Once the translation process is complete, you are ready to open the design in the Advanced Design System and simulate.

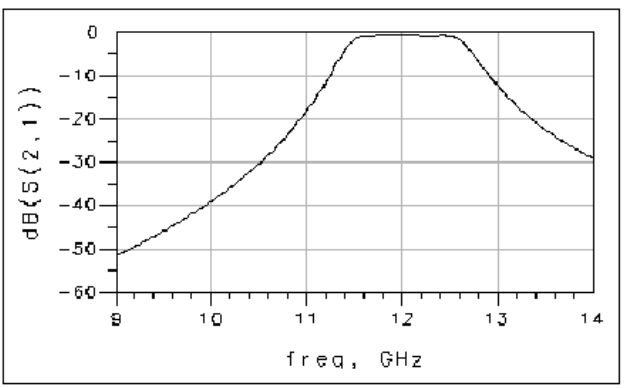
To simulate a translated design:

1. Launch ADS and open the new project containing the translated designs.

Hint Choose **View > Design Hierarchies** to see the hierarchical organization of the translated designs.

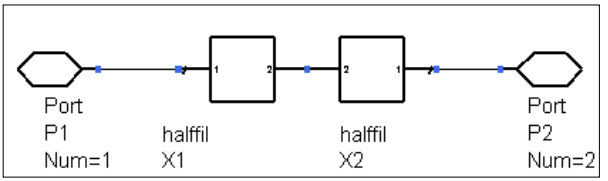


2. Open the **cmstpfil_tb** design.
3. Choose **Simulate** > **Simulate**. When the simulation is complete, choose **Window > New Data Display**.
4. Insert a rectangular plot and display S(2,1) in dB.

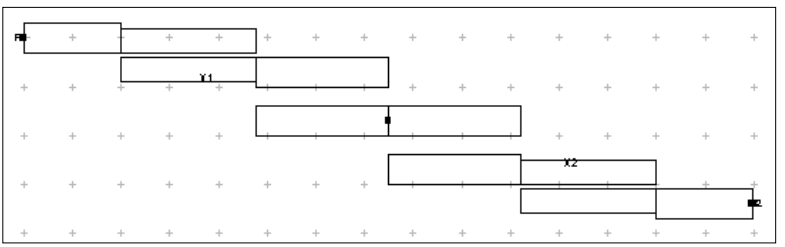


To view the network schematic and layout:

1. From the simulation design (*cmstpfil_tb*), push into *cmstpfil*.



2. Choose **Window** > **Layout** to display the *cmstpfil* layout.



Appendix A: Series IV Items Not Translated

These tables list various types of Series IV items that are not translated because no equivalent exists in ADS. All components not listed here are translated to ADS components or otherwise incorporated in the translated designs. The *Notes* column provides information about the changes in ADS. The file that controls how Series IV components are mapped to ADS components is *$HPEES0F_DIR/config/s4toads.rul*.

Schematic Items		
Library Group	**Exceptions**	**Notes**
Component and Ideal Elements	OPEN DSLTUNE2	No ADS equivalents for these items
Data File Elements	NWAS1P NWAS2P	The instrument server provides similar functionality. Refer to the manual, Using Instruments with ADS.
Measurement-related Elements	TP OSCTEST	The TP is translated as a named node. The TP instance name is used as the node name for the node to which the TP was connected. Because OSCTEST is a two-terminal device in ADS, components connected to pin 3 and 4 are not connected.
Microstrip Elements	EMPORT	Obsolete. Formerly for use with designs exported to Sonnet.

Test Items		
Library Group	**Exceptions**	**Notes**
Optimization/DOE/Yield Controls	DOE MEAS_HIST	No ADS equivalents for these items
Noise Measurements	INOISEQ VNOISEQ	No ADS equivalents for these items
Nonlinear Measurements	OSCAMNZ OSCNSPEC OSCPHNZ	No ADS equivalents for these items

Test Items (continued)		
Library Group	**Exceptions**	**Notes**
Simulation Controls	CABSTOL CONTOURS GMIN HFLIST IDMAX INITNH IRRANGE MAXTRTOT NOISE2P OSCDAMP OSCTOL	ADS uses different simulators with different controls
Output Files		Series IV *.s2p* files can be read by ADS. With regard to other Series IV output files, the ADS *dataset* provides alternative functions.

Vendor Components		
Manufacturer	**Part**	**Notes**
Motorola	pb60_mot_MRF9511_eebjt2	Translates as an obsolete part. Replace it with one of the following vendor parts: pb_mot_MRF9511L_19921101 or pb_mot_MRF951_19911008
Note: Only parts with the *60* designation (from Series IV 6.x) are translated; parts with a *40* or *50* designation (from Series IV 4.0, 5.0) are not translated.		

Miscellaneous Items	
Series IV Item	**Notes**
OmniSys components	These components are not translated, but they will be represented in the translated design as deactivated components. They will appear as red boxes labeled with the component name. Delete each of these components and replace them with the current ADS equivalent component through the Component Library.
Momentum Radiation Pattern definitions	Recalculate in ADS using Post Processing > Radiation Pattern from the Momentum menu. You will now have access to 3D visualization.

Miscellaneous Items (continued)	
Series IV Item	**Notes**
Momentum substrate database	Substrate must be recalculated in ADS
User-defined components	Refer to the section, Chapter 4, Updating User-Defined Models. *
Data display files (*.gra*)	Create new data displays in ADS after simulating. For details, refer to the Data Display manual.
* Designs that reference user-defined models cannot be translated until the user-defined models have been updated to ADS models (or linked/compiled with the ADS simulator).	

Appendix B: Nonlinear Model and Component Changes from Series IV to ADS

This appendix describes the basic differences between the nonlinear device models supplied in Series IV and their equivalents in ADS. The final table in the appendix shows the differences in the equation-based nonlinear components.

The affected nonlinear device models are:

These tables list the ADS equivalents for the Series IV model/device parameters and show how to achieve similar modeling effects. For example, when using the Curtice-Quadratic GaAsFET Model (Curtice2_Model) in ADS 2001, you need to assign a non-zero value to Cgs and set Gscap = 2 to achieve the same modeling effects you did in Series IV when assigning a non-zero value to CGSO. Both of these model the gate-source junction capacitance with a diode junction capacitance model.

The affected equation-based nonlinear components (in Table B-11) are:

- NonlinCCVS
- NonlinVCVS

PN-Junction Diode Model

Table B-1 describes the differences between the Series IV PN-Junction Diode Model (DIODEM) and its ADS equivalent (Diode_Model)

Table B-1. PN-Junction Diode Model (Diode_Model)

Series IV		**Advanced Design System 2001**	
Model Parameter	**Description**	**Model Parameter**	**Description**
None	Modeled by IDMAX	Imax=1000	Explosion current beyond which the diode junction current in the device is linearized Same as Series IV's IDMAX in a Simulation Control item. Default value for Imax is 1.0A. Default value for IDMAX is 1000.
None		AllParams	Instance Name of Data Access Component For accessing file-based model parameter values
None	Not modeled	wBv wPmax	Parameters for specifying the maximum voltage and power allowed for the device In a transient simulation, if OverloadAlert=yes (in the Tran component), a warning will be issued for each device exceeding these values at each time point. In a DC simulation, if GiveAllWarnings=yes (in the Options component), a warning will be issued for each device exceeding these values at the DC operating point.

Bipolar Transistor Model

Table B-2 describes the differences between the Series IV Bipolar Transistor Model (BJTM) and its ADS equivalent (BJT_Model).

Table B-2. Bipolar Transistor Model (BJT_Model)

Series IV		**Advanced Design System 2001**	
Model Parameter	**Description**	**Model Parameter**	**Description**
None	Modeled by IDMAX	Imax=1000	Explosion current beyond which the p-n junction currents in the device are linearized Same as Series IV's IDMAX in a Simulation Control item. Default value for Imax is 1.0A. Default value for IDMAX is 1000A.
None	†	Approxqb = yes	Use approximation for Qb vs. early voltage
None	†	Lateral = no	Vertical substrate geometry If set to Yes, the substrate junction is connected to the internal base and not the collector as in the vertical device
None	†	RbModel = Libra	Select Libra (Spice) base resistance equation The default is MDS, which uses MDS base resistance equation
None	Not modeled	wVsubfwd wBvsub wBvbe wBvbc wVbcfwd wIbmax wIcmax wPmax	Parameters for specifying the maximum voltages, currents, and power allowed for the device In a transient simulation, if OverloadAlert=yes (in the Tran component), a warning will be issued for each device exceeding these values at each time point. In a DC simulation, if GiveAllWarnings=yes (in the Options component), a warning will be issued for each device exceeding these values at the DC operating point.
None		AllParams	Instance Name of Data Access Component For accessing file-based model parameter values
† Use the value shown in the ADS Model Parameter column to achieve the same results as in Series IV (where this was modeled implicitly).			

EEsof Bipolar Transistor Model

Table B-3 describes the differences between the Series IV EEsof Bipolar Transistor Model (EEBJT2) and its ADS equivalent (EE_BJT2_Model).

Table B-3. EEsof Bipolar Transistor Model (EE_BJT2_Model)

Series IV		**Advanced Design System 2001**	
Model Parameter	**Description**	**Model Parameter**	**Description**
LB	Base inductance	None	Not modeled in ADS
LC	Collector inductance	None	Not modeled in ADS
LE	Emitter inductance	None	Not modeled in ADS
CXBE	External base-emitter fringing capacitance	None	Not modeled in ADS
CXBC	External base-collector fringing capacitance	None	Not modeled in ADS
CXCE	External collector-emitter fringing capacitance	None	Not modeled in ADS
None	Not modeled	wVsubfwd wBvsub wBvbe wBvbc wVbcfwd wIbmax wIcmax wPmax	Parameters for specifying the maximum voltages, currents, and power allowed for the device In a transient simulation, if OverloadAlert=yes (in the Tran component), a warning will be issued for each device exceeding these values at each time point. In a DC simulation, if GiveAllWarnings=yes (in the Options component), a warning will be issued for each device exceeding these values at the DC operating point.

Table B-3. EEsof Bipolar Transistor Model (EE_BJT2_Model) (continued)

Series IV		Advanced Design System 2001	
Model Parameter	**Description**	**Model Parameter**	**Description**
None		AllParams	Instance Name of Data Access Component For accessing file-based model parameter values
Device Parameter		**Device Parameter**	
None		Area = 1.0	Scaling factor. Default value is 1.0 The model is not intended for scaling with Area, however, ADS will scale if Area is set to anything other than 1.0. Be sure not to change the default value of 1.0.

Junction FET Model

Table B-4 describes the differences between the Series IV Junction FET Model (JFETM) and its ADS equivalent (JFET_Model)

Table B-4. Junction FET Model (JFET_Model)

Series IV		Advanced Design System 2001	
Model Parameter	**Description**	**Model Parameter**	**Description**
None	Modeled by IDMAX	Imax=1000	Explosion current beyond which the p-n junction currents in the device are linearized Same as Series IV's IDMAX in a Simulation Control item. Default value for Imax is 1.6A. Default value for IDMAX is 1000A.

Table B-4. Junction FET Model (JFET_Model)

Series IV		Advanced Design System 2001	
Model Parameter	**Description**	**Model Parameter**	**Description**
None		AllParams	Instance Name of Data Access Component For accessing file-based model parameter values
None	Not modeled	wBvgs wBvgd wBvds wIdsmax wPmax	Parameters for specifying the maximum voltages, currents, and power allowed for the device In a transient simulation, if OverloadAlert=yes (in the Tran component), a warning will be issued for each device exceeding these values at each time point. In a DC simulation, if GiveAllWarnings=yes (in the Options component), a warning will be issued for each device exceeding these values at the DC operating point.

MOSFET Models (Levels 1, 2, and 3)

Table B-5 describes the differences between the Series IV MOSFET Models (MOSLVL1, MOSLVL2, MOSLVL3) and their ADS equivalents (LEVEL1_Model, LEVEL2_Model, LEVEL3_Model).

Table B-5. MOSFET Models (LEVEL1_Model, LEVEL2_Model, LEVEL3_Model)

Series IV		Advanced Design System 2001	
Model Parameter	**Description**	**Model Parameter**	**Description**
None	Modeled by IDMAX	Imax=1000	Explosion current beyond which the p-n junction currents in the device are linearized Same as Series IV's IDMAX in a Simulation Control item. Default value for Imax is 1.6A. Default value for IDMAX is 1000A.
None		AllParams	Instance Name of Data Access Component For accessing file-based model parameter values

Table B-5. MOSFET Models (LEVEL1_Model, LEVEL2_Model,

Series IV		Advanced Design System 2001	
Model Parameter	**Description**	**Model Parameter**	**Description**
None	Not modeled		Parameters for specifying the maximum voltages, currents, and power allowed for the device
		wVsubfwd	Substrate junction forward bias
		wBvsub	Substrate junction reverse breakdown
		wBvg	Gate oxide breakdown voltage
		wBvds	Drain-source breakdown voltage
		wIdsmax	Maximum drain-source current
			In a transient simulation, if OverloadAlert=yes (in the Tran component), a warning will be issued for each device exceeding these values at each time point.
			In a DC simulation, if GiveAllWarnings=yes (in the Options component), a warning will be issued for each device exceeding these values at the DC operating point.
None	†	Capmod=3	Capacitance model selector
			To select the capacitance model used in Series IV (charge conserving first-order MOS charge model), set Capmod to 3. Default value is 1 (Meyer capacitance model).
† Use the value shown in the ADS Model Parameter column to achieve the same results as in Series IV (where this was modeled implicitly).			

Curtice-Quadratic GaAsFET Model

Table B-6 describes the differences between the Series IV Curtice-Quadratic GaAsFET Model (CURTICE2) and its ADS equivalent (Curtice2_Model)

Table B-6. Curtice-Quadratic GaAsFET Model (Curtice2_Model)

Series IV		**Advanced Design System 2001**	
Model Parameter	**Description**	**Model Parameter**	**Description**
CGS ≠ 0	Gate-source capacitance	Cgs ≠ 0 and Gscap = 1	Same as Series IV Linear model
CGD ≠ 0	Gate-drain capacitance	Cgd ≠ 0 and Gdcap = 1	Same as Series IV Linear model
CGSO ≠ 0 and CGS = 0	Zero-bias gate-source junction capacitance	Cgs ≠ 0 and Gscap = 2	Same as Series IV Diode junction capacitance model
CGDO ≠ 0 and CGD = 0	Zero-bias gate-drain junction capacitance	Cgd ≠ 0 and Gdcap = 2	Same as Series IV Diode junction capacitance model
RF ≠ 0	Gate-source linear forward conduction	Rf ≠ 0 and Gsfwd = 1	Same as Series IV Linear model
	Not modeled	Rf ≠ 0 and Gdfwd = 1	Gate-drain linear forward conduction Linear model
RF = 0	Gate-source diode conduction	Gsfwd = 2	Same as Series IV
	Gate-source diode reverse breakdown if VBR ≠ 0	Gsrev = 2 with Vbr, Ir, and Vjr	Gate-source diode reverse breakdown Series IV: $Igs = -IS \bullet \exp\left[\frac{Vsg - VBR}{N \bullet Vt}\right]$ ADS 1.5: $Igs = -Ir \bullet \left(\exp\left[\frac{Vsg - Vb}{Vjr}\right] - 1\right)$ where Vb = Vbr + R2 • Ids

Table B-6. Curtice-Quadratic GaAsFET Model (Curtice2_Model) (continued)

Series IV		Advanced Design System 2001	
Model Parameter	**Description**	**Model Parameter**	**Description**
R1 ≠ 0 VBR > 0	Gate-drain reverse linear breakdown	R1 ≠ 0 and Gdrev = 1	Same as Series IV Linear model
None	Not modeled	R1 ≠ 0 and Gsrev = 1	Gate-source reverse linear breakdown Linear model
R1 = 0	Gate-drain diode conduction	Gdfwd = 2	Same as Series IV
	Gate-drain diode reverse breakdown if VBR ≠ 0	Gdrev = 2 with Vbr, Ir, and Vjr	Gate-drain diode reverse breakdown Series IV: $Igd = -IS \bullet \exp\left[\frac{Vdg - VBR}{N \bullet Vt}\right]$ ADS 1.5: $Igd = -Ir \bullet \left(\exp\left[\frac{Vdg - Vb}{Vjr}\right] - 1\right)$ where Vb = Vbr + R2 • Ids
None	Not modeled	Rgd	Gate-drain resistance
None	Not modeled	Ld	Drain inductance Rd cannot equal zero
None	Not modeled	Lg	Gate inductance Rg cannot equal zero
None	Not modeled	Ls	Source inductance Rs cannot equal zero
None	Modeled by IDMAX	Imax = 1000	Explosion current beyond which the diode junction current in the device is linearized Same as Series IV's IDMAX in a Simulation Control itemL. Default value for Imax is 1.6A. Default value for IDMAX is 1000A.

Table B-6. Curtice-Quadratic GaAsFET Model (Curtice2_Model) (continued)

Series IV		**Advanced Design System 2001**	
Model Parameter	**Description**	**Model Parameter**	**Description**
None	Not modeled	Idstc ≠ 0 Betatce = 0	Use MDS temperature scaling equation for ids ids scaling factor is 1 + Idstc • (Temp – Tnom)
KF, AF, FFE	For noise model	R, P, C, Fnc	Same as Series IV The Series IV noise model has been replaced by the MDS noise model. Refer to the manual for details.
None	†	Taumdl = yes	Model ids transit time delay Set to *Yes* to use Series IV equations; Set to *No* to use MDS equations. This parameter affects transient analysis only.
None	Not modeled	wVgfwd wBvgs wBvgd wBvds wIdsmax wPmax	Parameters for specifying the maximum voltages, currents, and power allowed for the device In a transient simulation, if OverloadAlert=yes (in the Tran component), a warning will be issued for each device exceeding these values at each time point. In a DC simulation, if GiveAllWarnings=yes (in the Options component), a warning will be issued for each device exceeding these values at the DC operating point.
		AllParams	Instance Name of Data Access Component For accessing file-based model parameter values
† Use the value shown in the ADS Model Parameter column to achieve the same results as in Series IV (where this was modeled implicitly).			

Advanced Curtice-Quadratic GaAsFET Model

Table B-7 describes the differences between the Series IV Curtice-Quadratic GaAsFET Model (CURTICE2) and the ADS advanced model, Advanced_Curtice2_Model. (This advanced model was not available in Series IV). For differences in parameters not shown here, refer to the "Curtice-Quadratic GaAsFET Model" on page B-8.

Table B-7. Advanced Curtice-Quadratic GaAsFET Model (Advanced_Curtice2_Model)

Series IV		**Advanced Design System 2001**	
Model Parameter	**Description**	**Model Parameter**	**Description**
None	Not modeled	Ucrit	Critical field for mobility degradation†
None	Not modeled	Vgexp	Vgs – Vto exponent†
None	Not modeled	Gamds	Effective pinch-off combined with Vds†
		†These parameters modify Vto, Beta, and $(Vgs - Vto)^2$ in Curtice-Quadratic GaAsFET's Ids equation: $Vto_{NEW} = Vto + Gamds \bullet Vds$ $Beta_{NEW} = \frac{Beta}{(1 + (Vgs - Vto_{NEW}) \bullet Ucrit)}$ $(Vgs - Vto)^2 \leftarrow (Vgs - Vto_{NEW})^{Vgexp}$ When Ucrit ≠ 0, the above equations are in use and the temperature coefficients Vtotc and Betatc are disabled.	
RIN	Channel resistance	Rgs	Same as Series IV

Curtice-Cubic GaAsFET Model

Table B-8 describes the differences between the Series IV Curtice-Cubic GaAsFET Model (CURTICE3) and its ADS equivalent (Curtice3_Model)

Table B-8. Curtice-Cubic GaAsFET Model (Curtice3_Model)

Series IV		**Advanced Design System 2001**	
Model Parameter	**Description**	**Model Parameter**	**Description**
BETA	Coefficient for pinch-off change with respect to Vds	Beta2	Same as Series IV
VBO	Gate junction reverse bias breakdown	Vbr	Same as Series IV
CGS ≠ 0	Gate-source capacitance	Cgs ≠ 0 and Gscap = 1	Same as Series IV Linear model
CGD ≠ 0	Gate-drain capacitance	Cgd ≠ 0 and Gdcap = 1	Same as Series IV Linear model
CGSO ≠ 0 and CGS = 0	Zero-bias gate-source junction capacitance	Cgs ≠ 0 and Gscap = 2	Same as Series IV Diode junction capacitance model
CGDO ≠ 0 and CGD = 0	Zero-bias gate-drain junction capacitance	Cgd ≠ 0 and Gdcap = 2	Same as Series IV Diode junction capacitance model
RF ≠ 0	Gate-source linear forward conduction	Rf ≠ 0 and Gsfwd = 1	Same as Series IV Linear model
	Not modeled	Rf ≠ 0 and Gdfwd = 1	Gate-drain linear forward conduction Linear model

Table B-8. Curtice-Cubic GaAsFET Model (Curtice3_Model) (continued)

Series IV		**Advanced Design System 2001**	
Model Parameter	**Description**	**Model Parameter**	**Description**
RF = 0	Gate-source diode conduction	Gsfwd = 2	Same as Series IV
	Not modeled	Gsrev = 2 with Vbr, Ir, and Vjr	Gate-source diode reverse breakdown $Igs = -Ir \bullet \left(\exp\left[\frac{Vsg - Vb}{Vjr}\right] - 1\right)$ where Vb = Vbr + R2 • Ids
R1 ≠ 0 VBO > 0	Gate-drain reverse linear breakdown	R1 ≠ 0 and Gdrev = 1	Same as Series IV Linear model
	Not modeled	R1 ≠ 0 and Gsrev = 1	Gate-source reverse linear breakdown Linear model
None	Not modeled	Gdfwd = 2	Gate-drain diode conduction
		Gdrev = 2 with Vbr, Ir, and Vjr	Gate-drain diode reverse breakdown $Igd = -Ir \bullet \left(\exp\left[\frac{Vdg - Vb}{Vjr}\right] - 1\right)$ where Vb = Vbr + R2 • Ids
None	Not modeled	Rds0 ≠ 0	dc conductance at Vgs = 0 Add to Ids: $\frac{\tanh(Gamma \bullet Vds) \bullet (Vds - Vdsdc}{Rds0}$
CRF = 0 RDS ≠ 0	Additional output resistance for RF operation	Crf = 0 Rds ≠ 0	Rds – Crf is not modeled In Series IV, add to Ids: $\frac{(Vds - VdsDC)}{Rds}$ where VdsDC is Vds at DC bias solution after DC analysis
None	Not modeled	A5	Time delay proportionality constant for Vds delay = Tau + A5 • Vds
None	Not modeled	Rgd	Gate-drain resistance

Table B-8. Curtice-Cubic GaAsFET Model (Curtice3_Model) (continued)

Series IV		Advanced Design System 2001	
Model Parameter	**Description**	**Model Parameter**	**Description**
None	Not modeled	Ld	Drain inductance Rd cannot equal zero
None	Not modeled	Lg	Gate inductance Rg cannot equal zero
None	Not modeled	Ls	Source inductance Rs cannot equal zero
None	Modeled by IDMAX	Imax = 1000	Explosion current beyond which the diode junction current in the device is linearized Same as Series IV's IDMAX in a Simulation Control item. Default value for Imax is 1.6A Default value for IDMAX is 1000A
None	Not modeled	Idstc ≠ 0 Betatce = 0	Use MDS temperature scaling equation for ids ids scaling factor is 1 + Idstc • (Temp − Tnom)
KF, AF, FFE	For noise model	R, P, C, Fnc	Same as Series IV The Series IV noise model has been replaced by the MDS noise model. Refer to the manual for details.
None	†	Taumdl = yes	Model ids transit time delay Set to *Yes* to use Series IV equations; Set to *No* to use MDS equations. This parameter affects transient analysis only.

Table B-8. Curtice-Cubic GaAsFET Model (Curtice3_Model) (continued)

Series IV		Advanced Design System 2001	
Model Parameter	**Description**	**Model Parameter**	**Description**
None	Not modeled	wVgfwd wBvgs wBvgd wBvds wIdsmax wPmax	Parameters for specifying the maximum voltages, currents, and power allowed for the device In a transient simulation, if OverloadAlert=yes (in the Tran component), a warning will be issued for each device exceeding these values at each time point. In a DC simulation, if GiveAllWarnings=yes (in the Options component), a warning will be issued for each device exceeding these values at the DC operating point.
		AllParams	Instance Name of Data Access Component For accessing file-based model parameter values
† Use the value shown in the ADS Model Parameter column to achieve the same results as in Series IV (where this was modeled implicitly).			

Statz (Raytheon) GaAsFET Model

Table B-9 describes the differences between the Series IV Statz (Raytheon) GaAsFET Model (STATZ) and its ADS equivalent (Statz_Model)

Table B-9. Statz (Raytheon) GaAsFET Model (Statz_Model)

Series IV		Advanced Design System 2001	
Model Parameter	**Description**	**Model Parameter**	**Description**
THETA	Controls Ids – Vgs characteristic transition from quadratic to linear behavior	B	Same as Series IV
CGS ≠ 0	Gate-source capacitance	Cgs ≠ 0 and Gscap = 1	Same as Series IV Linear model

Table B-9. Statz (Raytheon) GaAsFET Model (Statz_Model) (continued)

Series IV		**Advanced Design System 2001**	
Model Parameter	**Description**	**Model Parameter**	**Description**
CGD ≠ 0	Gate-drain capacitance	Cgd ≠ 0 and Gdcap = 1	Same as Series IV Linear model
CGSO ≠ 0 and CGS = 0	Zero bias gate-source junction capacitance	Cgs ≠ 0 and Gscap = 6	Same as Series IV†
CGDO ≠ 0 and CGD = 0	Zero bias gate-drain junction capacitance	Cgd ≠ 0 and Gdcap = 6	Same as Series IV†
		†Currently, none of the available capacitance models in ADS 1.5 are fully compatible. Gscap = Gdcap = 6 will use the Statz charge-based model. The charge equation is the same as in Series IV, however, the partitions of the gate charge to gate-source and gate-drain are different.	
None		Vmax	Maximum junction voltage before capacitance limiting Series IV: Internal Vmax =FC • VBI ADS: Internal Vmax =Min(FC • VBI, Vmax)
None	Not modeled	Tqm	Junction capacitance temperature coefficient $C_{TEMP} = C \bullet (1 + Tqm \bullet (0.004 \bullet (Temp - Tnom) + (1 - Vbi_{TEMP}/Vbi)))$ where Vbi_{TEMP} is the temperature adjusted Vbi, C_{TEMP} is either Cgs_{TEMP} or Cgd_{TEMP}, and C is either Cgs or Cgd
None	Gate-source diode conduction	Gsfwd = 2	Same as Series IV In Series IV, this is automatically computed
None	Gate-drain diode conduction	Gdfwd = 2	Same as Series IV In Series IV, this is automatically computed

Table B-9. Statz (Raytheon) GaAsFET Model (Statz_Model) (continued)

Series IV		**Advanced Design System 2001**	
Model Parameter	**Description**	**Model Parameter**	**Description**
VBR ≠ 0	Gate-source reverse breakdown	Gsrev = 2 with Vbr, Ir, Vjr	Same as Series IV Series IV: $Igs = -IS \bullet \exp\left[\frac{Vsg - VBR}{N \bullet Vt}\right]$ ADS 1.5: $Igs = -Ir \bullet \left(\exp\left[\frac{Vsg - Vbr}{Vjr}\right] - 1\right)$
	Gate-drain reverse breakdown	Gdrev = 2 with Vbr, Ir, Vjr	Same as Series IV Series IV: $Igd = -IS \bullet \exp\left[\frac{Vdg - VBR}{N \bullet Vt}\right]$ ADS 1.5: $Igd = -Ir \bullet \left(\exp\left[\frac{Vdg - Vbr}{Vjr}\right] - 1\right)$
None	Not modeled	Rgd	Gate-drain resistance
None	Not modeled	Ld	Drain inductance Rd cannot equal zero
None	Not modeled	Lg	Gate inductance Rg cannot equal zero
None	Not modeled	Ls	Source inductance Rs cannot equal zero
None	Modeled by IDMAX	Imax = 1000	Explosion current beyond which the diode junction current in the device is linearized Same as Series IV's IDMAX in a Simulation Control item. Default value for Imax is 1.6A. Default value for IDMAX is 1000A.

Table B-9. Statz (Raytheon) GaAsFET Model (Statz_Model) (continued)

Series IV		Advanced Design System 2001	
Model Parameter	**Description**	**Model Parameter**	**Description**
None	Not modeled	Idstc ≠ 0 Betatce = 0	Use MDS temperature scaling equation for ids ids scaling factor is 1 + Idstc • (Temp – Tnom)
KF, AF, FFE	For noise model	R, P, C, Fnc	Same as Series IV The Series IV noise model has been replaced by the MDS noise model. Refer to the manual for details.
None	†	Taumdl = yes	Model ids transit time delay Set to *Yes* to use Series IV equations; Set to *No* to use MDS equations. This parameter affects transient analysis only.
None	Not modeled	wVgfwd wBvgs wBvgd wBvds wIdsmax wPmax	Parameters for specifying the maximum voltages, currents, and power allowed for the device In a transient simulation, if OverloadAlert=yes (in the Tran component), a warning will be issued for each device exceeding these values at each time point. In a DC simulation, if GiveAllWarnings=yes (in the Options component), a warning will be issued for each device exceeding these values at the DC operating point.
		AllParams	Instance Name of Data Access Component For accessing file-based model parameter values
† Use the value shown in the ADS Model Parameter column to achieve the same results as in Series IV (where this was modeled implicitly).			

TriQuint Scalable Nonlinear GaAsFET Model

Table B-10 describes the differences between the Series IV TriQuint Scalable Nonlinear GaAsFET Model (EETOM1) and its ADS equivalent (TOM_Model). Note:

The TOM and EETOM models of Series IV have been merged into one TOM model in ADS 1.5.

Table B-10. TriQuint Scalable Nonlinear GaAsFET Model (TOM_Model)

Series IV		**Advanced Design System 2001**	
Model Parameter	**Description**	**Model Parameter**	**Description**
GAMMADC	DC drain pull coefficient	Tqgamma	Same as Series IV
GAMMA	AC drain pull coefficient	TqgammaAC	Same as Series IV (available in ADS)
DELTA	Output feedback coefficient	Tqdelta	Same as Series IV
CGSO ≠ 0 and CGS = 0	Zero-bias gate-source capacitance	Cgs ≠ 0 and Gscap = 5	Same as Series IV
CGDO ≠ 0 and CGD = 0	Zero-bias gate-drain capacitance	Cgd ≠ 0 and Gdcap = 5	Same as Series IV
RDB	Dispersion source output impedance	Rdb	Same as Series IV
CBS	Dispersion source capacitance	Cbs	Same as Series IV
None	Use grading coefficient parameter M, no separate parameter	Tqm	Temperature coefficient for TriQuint junction capacitance (default = 0.2) In ADS: $Cgs(Temp) = Cgs[1 + Tqm \bullet [4.0 \bullet 10^{-4} \bullet (Temp - Tnom) + (1 - Vbi(Temp)/Vbi)]]$ $Cgd(Temp) = Cgd[1 + Tqm \bullet [4.0 \bullet 10^{-4} \bullet (Temp - Tnom) + (1 - Vbi(Temp)/Vbi)]]$
None		Vmax	Maximum junction voltage before capacitance limiting In Series IV: Internal Vmax = FC • VBI In ADS: Internal Vmax = Min(Fc • Vbi, Vmax)
None	Not modeled	Trg1	Linear temperature coefficient for Rg

Table B-10. TriQuint Scalable Nonlinear GaAsFET Model (TOM_Model) (continued)

Series IV		**Advanced Design System 2001**	
Model Parameter	**Description**	**Model Parameter**	**Description**
KF, AF, FFE	For noise model	R, P, C, Fnc	Same as Series IV The Series IV noise model has been replaced by the MDS noise model. Refer to the manual for details.
None	†	Taumdl = yes	Model ids transit time delay Set to *Yes* to use Series IV equations; Set to *No* to use MDS equations. This parameter affects transient analysis only.
None	Not modeled	wVgfwd wBvgs wBvgd wBvds wIdsmax wPmax	Parameters for specifying the maximum voltages, currents, and power allowed for the device Comment: In a transient simulation, if OverloadAlert=yes (in the Tran component), a warning will be issued for each device exceeding these values at each time point. In a DC simulation, if GiveAllWarnings=yes (in the Options component), a warning will be issued for each device exceeding these values at the DC operating point.
		AllParams	Instance Name of Data Access Component For accessing file-based model parameter values
Device Parameter		Device Parameter	
UGW	New unit gate width (EETOM1)	W	Same as Series IV
TOP	Device operating temperature in °C (EETOM1)	Temp	Same as Series IV
† Use the value shown in the ADS Model Parameter column to achieve the same results as in Series IV (where this was modeled implicitly).			

Equation-Based Nonlinear Components

Table B-11 shows the minor differences between the Series IV equation-based nonlinear components and their ADS equivalents.

Table B-11. Equation-Based Nonlinear Components

Series IV	**Advanced Design System 2001**	
Component	**Component**	**Comment**
NLCCVS	NonlinCCVS	The polarity of the output voltage source in ADS is opposite of what it was in Series IV.
NLVCVS	NonlinVCVS	The polarity of the output voltage source in ADS is opposite of what it was in Series IV.

Appendix C: Translator Customization

The translation of components is accomplished using a database of rules that maps Series IV components to ADS components parameter-by-parameter. Standard components are mapped through the supplied rules file, *s4toads.rul*, but you need to create a custom rules file for custom parts. When translating designs or projects, the translator first looks for a custom rules database and then for the supplied rules database. Custom translation rules take precedence over supplied rules.

Writing Custom Translation Rules

Rules files can be created using any text editor. An empty line or any line beginning with the pound symbol (#) is ignored. Examples of rules used by the translator can be found in the *$HPEESOF_DIR/config/*.rul* files. Translation rules use the following syntax and should be on a single line of the rules file:

```
OldName|ADSName|ParmRule|DefRule|PinRule|NameRule|DelFlag|MappedParmRule|
```

Field	**Description**
OldName	Series IV component name
ADSName	ADS component name
ParmRule	Parameter mapping AEL function name (optional)
DefRule	Component definition mapping rule (obsolete)
PinRule SrcPinNum SrcPinName DestPinNum DestPinName	Pin mapping rule list (optional) Old pin number Old pin name (obsolete) Destination pin number Destination pin name (obsolete)
NameRule	Name mapping AEL function name (optional)
DelFlag	Delete flag (TRUE or FALSE)
Note that when any of the above fields is not used (obsolete or optional) the field separator "|" must still be included.	
MappedParmRule ADSParmName ParmName DataItemName DataItemParmName DataItemParmDefForm	Mapped parameter rule list (optional) ADS parameter name Series IV parameter name Series IV Data Item Name Series IV Data Item parameter This parameter is no longer used

Example:

```
XFER|Transformer| | | | |FALSE| |
```

In this example, a Series IV component named XFER is translated to an ADS component named Transformer. All parameters that have the same name in Series IV as they do in ADS are copied to the new component. A simple rule of this form will translate most custom parts.

Example:

```
XFER| | | | | |TRUE| |
```

In this example (setting the Delete Flag to TRUE), the Series IV component will not be translated. The status messages produced during translation will indicate that the component is not translatable and has been removed from the design.

If the names of component parameters need to be changed during the translation, you need to use the MappedParmRule field. The syntax for this field is:

```
ADSParmName,[@|&|%]ParmName,DataItemName,DataItemParmName,
DataItemParmDefForm,
```

Field, MappedParmRule	Description
ADSParmName	ADS Component Parameter Name
ParmName @ - Followed by a default value of the parameter. This allows a new ADS parameter to have a default value rather than copying a value from the SIV component & - The parameter value of Series IV component will be quoted as a string. This is normally needed in ADS for items that are not numeric such as the name of a substrate or a filename. % - The case of the specified SIV parameter name is not significant (e.g., Cond and COND refer to the same parameter)	Series IV Component Parameter Name
DataItemName	Series IV Data Item from which to retrieve the named parameter
DataItemParmName	Series IV Data Item Parameter from which to retrieve the value
DataItemParmDefForm	This parameter is no longer used.

Example:

```
GYR|Gyrator|  |  |  |False|Ratio,R,  ,  ,  ,;|
```

In this example, the rule copies the value from the old name (R) to the new name (Ratio). The third through fifth parameters are not used.

Example:

```
MONOPOLE|AntLoad| | | | |FALSE|AntType,@MONOPOLE, , , ,;RatioLR,LR, , ,
,;Length,L, , , ,;|
```

In this example, a Series IV component called MONOPOLE is translated to a more general ADS component named AntLoad. The parameter AntType is given a default value of "MONOPOLE" by using the "@".

Retrieving Parameter Values from Series IV Data Items

Because ADS does not support the Series IV Data Item, you may want to copy a parameter value of a Series IV Data Item to the appropriate ADS component parameter using the DataItemName and DataItemParmName fields. For example, you can retrieve the value of the TEMP parameter of the Series IV TEMP Data Item—using these fields—and copy it to the ADS parameter Temp, for a given component.

Example:

```
RES|R|  |   |  |FALSE|Temp,TEMP,TEMP,TEMP, ;|
```

In this example, the Series IV component RES is translated to the ADS component R and all parameters that have the same name are copied. The MappedParamRule says to copy the value of the TEMP parameter of the Series IV TEMP Data Item to the ADS parameter Temp (for this component).

Setting Parameter Values Using AEL Functions

The translator can invoke an AEL function during the translation. To use this feature, you specify the name of an AEL function in the ParmRule field. This enables you to extend the translation capabilities, such as computing numerical values from other parameters.

Four pieces of information are combined as a list and sent to the AEL function, which will return an updated parameter list to the translator. The parameter list sent to the AEL function will already have parameter values for all parameters that have the same name in the Series IV component that they have in the ADS component. It will

also have any parameters copied that are specified by a MappedParmRule. The translator will use the updated parameter list to modify the replaced instance.

Input Parameter:

```
list(list(sivParmList,    /*Series component parameter list*/
    adsParmList),   /*ADS component parameter list*/
    instH,          /*New ADS instance handle*/
    sivItemName)    /*Series IV component Name*/
```

Output Parameter:

```
adsParmList - Updated ADS component parameter list
```

The syntax for sivParmList and adsParmList is shown next.

```
sivParmList =
```

or

```
adsParmList =

list(list(Name1, Form1, Value1, UnitString1, UnitCode1),
    list(Name2, Form2, Value2, UnitString2, UnitCode2),
    ...
    list(NameN, FormN, ValueN, UnitStringN, UnitCodeN))
```

where

Name	Name of parameter
Form	Form of the current parameter
Value	Value of the parameter (with unit string)
UnitString	Unit string of the current parameter
UnitCode	Unit code of the parameter

Consider the following AEL function:

```
defun merullib_SUBSTRATE ()
{
    decl newParmH, tmpStr, newParmName;
    decl oldParmList = car(car(arg_list()));
    decl newParmList = car(cdr(car(arg_list())));
    decl j, i = listlen(newParmList);
    decl rhsH;

    if (oldParmList != NULL)
    {
      for(j=0; j<i; j++)
      {
          tmpStr = "";
          newParmH = nth(j, newParmList);
          newParmName = car(newParmH);

          if (!strcmp("Cond", newParmName))
          {
              decl rhoH;
              //
              //  get parameter RHO
              //
              tmpStr = merul_extract_parm(oldParmList, "RHO", NULL, NULL, NULL);
                                    if( tmpStr )
              {
                  rhoH = car(merul_plib_parse(tmpStr, nth(MERUL_PARM_UNIT_CODE,
                  newParmH)));
                   //
                  //  construct parameter
                  //
                  if (rhoH || strlen (rhoH) !=0)
                  {
                                           if (val (rhoH) != 0)
                          tmpStr = identify_value(1.0E+50/val (rhoH));
                      else
                          tmpStr = identify_value(1.0E+50);
                  }
                  else
                      tmpStr = identify_value(1.0E+50);
              }
          }
          else
          if (!strcmp("Mur", newParmName))
          {
              decl tmpParmList = meutil_get_dataitem_parmlist( TRUE, "PERM", FALSE);
              tmpStr = merul_extract_parm(tmpParmList, "MUR", NULL, NULL, NULL);
          }
          else
          if (!strcmp("TanD", newParmName))
          {
              decl tmpParmList = meutil_get_dataitem_parmlist( TRUE, "TAND", FALSE);
              tmpStr = merul_extract_parm(tmpParmList, "TAND", NULL, NULL, NULL);
          }
```

```
          if(tmpStr)
            {
                newParmH    = repla(newParmH, tmpStr, MERUL_PARM_VALUE);
                newParmList = repla(newParmList, newParmH, j);
            }
        }
    }
    return newParmList;
}
*/
```

This example illustrates most features of a typical ParmRule AEL function.

When this AEL function runs, all parameters with the same name in both the Series IV and the ADS components, have already been copied to the ADS component. This AEL function only needs to modify parameters that need special handling.

The following is a description of what occurs when the function runs:

- On the fourth and fifth lines, it gets the oldParmList and newParmList for the Series IV and ADS components.
- The next line sets "i" to the length of the newParmList.
- On line 11 a "for" loop is started that loops through the parameters from the newParmList.
- The variable "newParmName" is assigned the name of the next parameter of the new component.
- A series of conditional "if" statements looks for specific parameter names and does special handling for these parameters. Within the "if" statement that selects "Cond" the function merul_extract_parm (line 23) is used to get the value of the "RHO" parameter from the old component.
- The third and forth parameters of merul_extract_parm are DataItemName and DataItemParmName, as described previously.
- The term "tmpStr" is a string containing the old parameter.
- The term rhoH is the numeric value of the parameter including applying the unit associated with this parameter.
- The lines starting at line 30 compute a string value for tmpStr based on the numeric value of the RHO parameter. Near the end of the function, if tmpStr had a value, a newParmH is created for tmpStr and it is used to update the parameter list for the new component.

Mapping Component Pin Changes

If the pin connections of the old component and the new component are not the same, you can create a PinRule to interchange the pin numbers using the following sub-fields:

SrcPinNum,SrcPinName,DestPinNum,DestPinName,;

where

SrcPinNum	Old pin number (optional)
SrcPinName	Old pin name (not used)
DestPinNum	Destination pin number
DestPinName	Destination pin name (not used)

Note that when any of the above fields is not used (or is optional), the field separator, a comma (,) must still be used. A semicolon (;) must be used at the end, and multiple PinRules can be appended together to map several pins, with each PinRule being separated by a semicolon (;).

Note If DestPinNum is specified but SrcPinNum is not, the DestPinNum of the new instance will be grounded.

Example:

```
Neg1|Deembed1|  |  | , ,2, ,;|  |FALSE| |
```

In this example, pin 2 of the ADS Deembed1 component would be grounded.

Example:

```
PCCROS|PCCROS|  |  |2, ,4, ,;3, ,2, ,;4, ,3, ,;|  |FALSE| |
```

This example contain 3 PinRules. Series IV pin 2 is mapped to ADS pin 4, Series IV pin 3 is mapped to ADS pin 2, and Series IV pin 4 maps to ADS pin 3.

Mapping to a Component Based on Parameter Specifics

If you want to map a Series IV component to an ADS component based on a parameter from the Series IV component, you can specify an AEL function in the NameRule field.

Example:

```
defun merullib_name_P2D()
{
    decl oldParmList = car(car(arg_list()));
    if (listlen(oldParmList) == 1)
    {
        return "DataAccessComponent";
    }
    else
    {
        return "AmplifierP2D";
    }
}
```

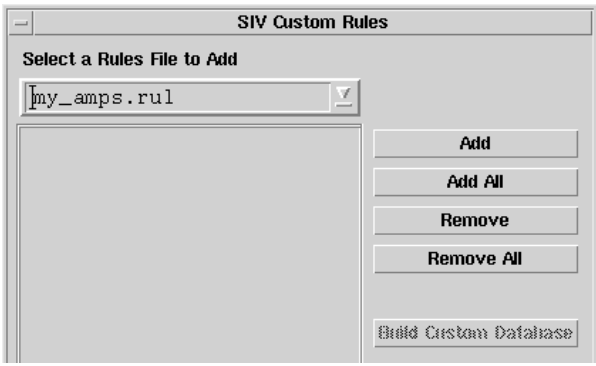
This example looks at the number of parameters of the Series IV component. If there is one parameter, the name "DataAccessComponent" is returned, otherwise "AmplifierP2D" is returned.

Using a Customized Rules Files

To use custom rules, you need to create one or more rules files and place them in your *$HOME/hpeesof/config* (UNIX) or *%HOME%\hpeesof\config* (PC) directory. These files must end with the *.rul* suffix and must be compiled into a rules database file (*meruls_siv_custom.db*).

To compile your custom rules file(s) do the following:

1. Launch the translator as described in “Importing Designs” on page 2-2.
2. Click **Custom Rules** and a dialog box appears. If any *.rul* files are found in the local config directory, they will appear on the drop-down list.



3. Select the desired rules file from the drop-down list and click **Add**. Alternatively, click **Add All** to merge all rules files into one custom database.

Hint To modify the entries in the list box, highlight an individual one and click **Remove**. Click **Remove All** to clear the box and start over.

---

4. Once the list box contains the desired filename(s), click **Build Custom Database**. The status panel at the bottom of the dialog box displays messages reporting the progress. When the process is complete, a message that says “The rules database has been successfully built” or something similar will appear.
5. Once the database has built successfully, you must select the option **Use Custom Database** for it to be used for the upcoming translation.
6. Click **OK** and proceed with the translation.

Customizing Translator Variables

The variables listed in Table C-1 can be customized as shown to modify the default behavior of the translator.

Table C-1. Series IV to ADS Translator Customization Variables

Variable/Description	Valid Values
MIGRATION_MOVE_ANNOTATION Controls the location of component annotation. TRUE causes annotation to be drawn in its original Series IV location; FALSE allows ADS to draw the annotation where it wants. (Default is TRUE) Example: MIGRATION_MOVE_ANNOTATION=TRUE	TRUE or FALSE
MIGRATION_TESTBENCH_TEMPLATE Controls the translation of test bench templates. TRUE means Series IV test bench templates (convolution_template_tb.dsn, dcbias_template_tb.dsn, etc.) are translated; FALSE means they will not be translated. (Default is FALSE) MIGRATION_TESTBENCH_TEMPLATE=FALSE	TRUE or FALSE

Appendix D: Batch Mode Translation

This section describes how to set up and perform SIV batch translation from the command line.

Note Translating SIV designs from a command line requires knowledge of how to set environment variables and work in a DOS window or UNIX shell. Therefore, this task should only be attempted by advanced users.

---

hpeesofme Setup

Prior to use, the *hpeesofme* program must be properly configured.

Windows

1. Open an MS DOS shell.
2. Set the $HPEESOF_DIR environment variable to your ADS installation directory. For example

   set HPEESOF_DIR=<ADS_install_dir>
3. Set your PATH environment variable to include the *$HPEESOF_DIR\bin* directory. For example

   set path=C:\ADS2001\bin;%path%
4. Set the appropriate library path for your operating system. For example

   set shlib_path=$shlib_path:$HPEESOF_DIR\lib\win

UNIX

1. Set your `$HPEESOF_DIR` environment variable to your ADS installation directory. For example, if using the Korn shell enter

   export HPEESOF_DIR=<ADS_install_dir>
2. Set your `PATH` environment variable to include the *$HPEESOF_DIR/bin* directory. For example, if using the Korn shell enter

   export PATH=$HPEESOF_DIR/bin:$PATH

3. Set the appropriate library path for your operating system. For HP-UX operating systems (i.e. hpux10 or hpux11), enter the following:

   SHLIB_PATH=$SHLIB_PATH:$HPEESOF_DIR/lib/hpux10

   or

   SHLIB_PATH=$SHLIB_PATH:$HPEESOF_DIR/lib/hpux11

   For SUN operating systems (i.e. sun4 or sun55), enter

   LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HPEESOF_DIR/lib/sun4

   or

   LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HPEESOF_DIR/lib/sun55

   For AIX operating systems (i.e.aix4), enter the following:

   LIBPATH=$LIBPATH:$HPEESOF_DIR/lib/aix4

Executing hpeesofme

The SIV translator is started using the following syntax:

```
hpeesofemx hpeesofme -me -nw -mep <projectDef> -mec <customRules> -env
<envFileName>
```

where:

-me	Forces the translation to start after bootup. Required
-nw	Invokes the program in the non-visual mode. Required
-mep <projectDef>	Specifies the project to migrate. Required
-mec <customRules>	Specifies the custom rules database (with path and extension). Optional
-env <envFileName>	Specifies the environment file to be used (usually *de_sim*). Required

Additional Option Information

-mep

This option has the following syntax:

“SrcPath|DefaultDsnName|0|DestPath|UnitPref”

where:

SrcPath	Source project/design path
DefaultDsnName	Default design name to be used for SIV translation
DestPath	Destination project path
UnitPref	Unit preference (0: mil, 1: millimeter, 2: micron)

Example

The following is an example of a Perl script capable of running multiple translations in batch mode:

```
# a script to run the Series IV translator from command line
#
@projects = (“proj1”, “proj2”);
$sivdir = “c:/users/toto/ExamplesSIV/tests”;
$adsdir = “c:/users/toto/adsmigration/sivmigrated”;
$customRules = “c:/users/toto/hpeesof/config/meruls_s4_custom.db”;

foreach (@projects)
{
print “\n\n********* translate $_ **********\n\n”;
$sivfile = “$sivdir/$_”.’_prj’;
$adsproj = “$adsdir/$_”.’_prj’;
system(“hpeesofemx hpeesofme -nw -me -mep \”$sivfile|defaults|0|$adsproj\”
-mec \”$customRules\” -env de_sim”);
system(“egrep -e ERROR -e WARNING -e Error -e Warning $adsproj/me_err.log”);
}
close LOGFILE;
```

Appendix E: Known Issues

Annotation Location

By default, component annotation is placed (relative to the symbol) in the same location as it was in Series IV. To allow the translator to move it (where applicable) to the default ADS location, change the MIGRATION_MOVE_ANNOTATION variable from TRUE to FALSE. For details, refer to "Customizing Translator Variables" on page C-10.

Artwork Macros

A large number of AEL functions changed between Series IV and ADS, however those typically used in artwork macros have changed very little. The main change is that the functions in ADS include '*de_*' in front of the old function name. For example, in Series IV there was a function, *draw_rect*, and in ADS that same function is *de_draw_rect*. While the old function names are still supported, we recommend that for any new macros you create, you use the new function names.

Your Series IV artwork macros should require very little change. One change that you must make however is to add the following line to the beginning of each file:

de_set_global_db_factor();

Node Names

The SIV translator allows the same name to be used for test points in different designs (e.g., a test bench and a sub-network). During translation to ADS, these test points are translated into named nodes. However, ADS does not allow the same name to be used for two different nodes, even if they are used in different designs in the hierarchy. When different designs with common test point names are translated into ADS, the named nodes need to be manually renamed so that each name is unique.

Port Rotation

After translating, you will notice that some ports are incorrectly rotated. This is due to the fact that ports do not have any rotation information in Series IV. You can rotate them if you like, but the design will simulate correctly as-is.

AEL Functions

The log Function

Series IV provided the log function

log()

which returned the natural logarithm. ADS has three log functions:

ln()	Returns the natural logarithm of an integer, real, or complex number
log() or log10()	Returns the base 10 logarithm of an integer or real number

Because the function log() is legitimate in ADS, the translator does not replace the Series IV function and your results will vary. To get the same results you did in Series IV, replace log() with ln().

Miscellaneous Functions

The following script can be used to search for functions that may need modification after migration. Save the text to a file (call it “check_for_funtions.pl”).

Execute this script either by typing

```
check_for_functions.pl
```

at the command line (if a perl executable is in your path) or by typing (shown here for a PC installation):

```
%HPEESOF_DIR%\tools\bin\perl check_for_functions.pl
```

where the HPEESOF_DIR path has been set to point to your installation of ADS.

```
# PERL Script "check_for_functions.pl"
#
# This function is provided as a convenience to help find possible code in a Series IV
# Senior model that may need changes to be compatible with ADS user-compiled models.
# The script finds functions whose definition has changed and for which the developer
# should check for possible unexpected results.
#
# Execute this script either by typing the name at the command line (if a perl executable is
# in your path) or by typing %HPEESOF_DIR%\tools\bin\perl check_for_functions.pl where the
# HPEESOF_DIR path has been set to point to your installation of ADS. The script only checks
# files in the current directory, so typically this script should be run from your project's
# networks directory.
#
# copyright Agilent Technologies, Inc. 2000

# Each of the following strings will be search for in the file list (additional strings
# can be added anywhere in this section):
```

```
# For most unit functions, ADS will return the value 1.0 and the changes of the units is
# taken care of on the parameter value scale factor:
  push @function_list, "get_funit";
  push @function_list, "get_runit";
  push @function_list, "get_gunit";
  push @function_list, "get_lunit";
  push @function_list, "get_cunit";
  push @function_list, "get_lenunit";
  push @function_list, "get_tunit";
  push @function_list, "get_angunit";
  push @function_list, "get_curunit";
  push @function_list, "get_volunit";
  push @function_list, "get_watt";

# Because parameter names and index positions may have changed, code using these
# functions should be checked carefully:
  push @function_list, "get_params";
  push @function_list, "get_user_inst";

# This function is mapped to send_info_to_scrn:
  push @function_list, "send_info_to_file";

  opendir CURRDIR, "." or die "Error: $!";
  @file_list = grep /\.[chCH]$/,readdir CURRDIR ;
  closedir CURRDIR;

  foreach $file (@file_list)
  {
      print $file;
      print ": ";
      open(CURRFILE, "<$file");
      $count = 0;
      $match_flag = 0;
      while (<CURRFILE>)
      {
$count++;
foreach $function (@function_list)
{
          if (/$function/)
  {
    if ($match_flag == 0)
          {
  print " Possible changes needed on lines:\n";
          }
            $match_flag = 1;
    print $file
    print "  ";
    print $count;
    print " ";
    print trim("$_");
    print "\n";
          }
}
      }
      if ($match_flag == 0)
      {
print "Changes probably not needed.\n";
      }
  }

sub trim {
```

```
my @out = @_;
for (@out)
{
  s/^\s+//;
  s/\s+$//;
}
return wantarray ? @out : $out[0];
}
```

Index