cādence®

AMS Designer in ADE Explorer FAQ

Product Version ICADVM20.1
October 2020

Contents

AMS Designer in ADE Explorer FAQ

This page contains the answers to customer's most frequently asked questions about the interface to the Spectre AMS Designer and Xcelium Mixed-Signal in ADE and ADE Explorer. For more information, see the *Virtuoso ADE Explorer User Guide*.

The frequently asked questions are organized under the following sections:

General FAQ on the AMS Designer Interface in ADE Explorer

Which AMS netlisters are available in ADE Explorer to netlist my mixed-signal design?

ADE Explorer provides the following three Verilog-AMS netlisters to run your mixed-signal design:

- AMS Unified Netlister (AMS UNL)
- AMS OSSN netlister
- AMS Cellview-Based netlister

The AMS UNL netlister is the default AMS netlister used for ADE Explorer. It is strongly recommended to use this netlister over other legacy netlisters. Any new user to AMS in ADE Explorer should use AMS UNL. The two legacy netlisters will be deprecated once legacy users fully migrate to AMS UNL.

As AMS UNL is the solution moving forward, this document will only mention AMS UNL flow aspects.

Which simulation mode is available in ADE Explorer to simulate the netlist created by the AMS UNL netlister flow?

If you create the netlist using the AMS UNL flow, ADE Explorer will simulate the design using the xrun executable from the AMS Designer simulator.

Does AMS Designer support multi-threading capability?

Yes, AMS Designer supports multi threading capability with Spectre solver. You can enable multithreading by passing the following options to `xmsim` or `xrun`:

```
xmsim -spectre_args +mt=<number_of_threads>
xrun -spectre_args +mt=2
xrun -spectre_args +mt    // Simulator automatically chooses the number of threads
```

If you are running APS solver from ADE Explorer, you can enable/disable multithreading using the *High-Performance Simulation Options* form under the *Setup* menu. You can choose from the following options:

- Disable - On selection, disables multithreading.
- Auto - On selection, enables multithreading and automatically chooses the number of threads.

- Manual - On selection, enables multithreading and allows you to specify the number of threads

Note: Currently, multithreading is available only on the linux and solaris platforms.

How to compile user defined connect modules?

In order to compile user defined connect modules, you need to do the following:

- Create 5x hierarchy for connect modules, as shown below:

  ```
  ncvlog -ams -use5x -work libName module_file
  ```

- Create a *`connectRules.il`* file under *`libname`*, as shown below:

  ```
  cd libName
  genConnRulesFile -lib $workLib */*/*.vams -destpath .
  ```

  All the connect modules in the *`connectRules.il`* file will be listed in the *Built-in rules* groupbox of ADE Explorer *Select Connect Rules* form.

Note: Ignore the second step if you do not want to display user-defined rules as built-in rules.

General FAQ on the AMS UNL Netlister

What is the AMS UNL flow?

UNL stands for Unified Netlister. It is the final AMS netlisting solution culminating many years of development and lessons learned from past AMS Netlisting flows. Its purpose is to support complex text natively in AMS designs of today. Also, it is meant to replace all previous AMS Virtuoso flows including SpectreVerilog, AMS Cell-based, and AMS OSSN with one "unified" solution supporting all aspects of complex AMS designs. It is still a partially OSS-based netlister utilizing the OA side of hierarchical netlisting.

What is UNL Verilog-AMS netlister?

The UNL Verilog-AMS netlister is a Verilog-AMS netlister integrated into ADE and ADE Explorer. It translates schematics and layout cellviews into a Verilog-AMS structural netlist for compilation, elaboration, and simulation.

How can I set the AMS UNL netlister in ADE Explorer?

On the ADE Explorer window, choose *Simulation - Netlist and Run Options*. On the resulting form, select *AMS Unified netlister with xrun*.

Alternatively, do one of the following:

- Add the following entry in your `.cdsinit` file:

  ```
  envSetVal("ams.envOpts" "netlisterMode" 'string "AMS-UNL")
  ```

- Add the following entry in your `.cdsenv` file:

  ```
  ams.envOpts netlisterMode string "AMS-UNL"
  ```

What is the name of the netlist file created by the UNL netlister, and where is it located?

The UNL netlister creates a `netlist.vams` file in the netlist directory.

Does the UNL netlister also create cellview-level netlists like the Cellview-based netlister?

No. The UNL netlister creates a single netlist of all the schematics. However, this netlist does not contain all the design information needed by the simulator. Several other files such as `textInputs` file, simulator control file and probe Tcl file that contain design information are also passed to the simulator.

Does the UNL netlister also need the implicit/explicit temporary library or writable design library?

No. The UNL netlister creates the netlist in the netlist directory and all compiled text and netlist in the xrun `INCA_libs` directory for every run. Hence, it does not need any external temporary library or writable design library.

Can I save OCEAN scripts for the UNL netlister and run them later?

Yes.

If I check-and-save a schematic in Virtuoso Schematic Editor (Composer), does it lead to running the UNL netlister, like the Cellview-based netlister?

No.

Like the amsdesigner command-line executable for the Cellview-based netlister, is there a command-line executable for the UNL netlister?

Yes.

The command-line executable for the UNL netlister is called `runams`, the same as it was for the AMS OSSN netlister. The use model and commands have not changed from the AMS OSSN flow.

FAQ on xrun

What is xrun?

`xrun` is an executable available in your IUS release. It allows you to compile, elaborate, and simulate your design using a single command. `xrun` is the replacement for `ncverilog`.

`xrun` can simulate Verilog, SystemVerilog, VHDL/VHDL-AMS, Verilog-AMS, Verilog-A, C, C++, SPICE, compiled object, dynamic library, and PSL files for Verilog/VHDL/SystemC.

`xrun` uses file extension to determine file type, and automatically compiles them using the appropriate compiler. After compilation, `xrun` automatically invokes `xmelab` to elaborate the design, and then `xmsim` to simulate.

What is the advantage of xrun over ncverilog?

`ncverilog` can simulate only Verilog and Verilog-AMS, whereas `xrun` can simulate Verilog, SystemVerilog, VHDL, Verilog-AMS, C, C++, SPICE, compiled object, dynamic library, and PSL files for Verilog, VHDL and SystemC.

What is the advantage of single step simulation (xrun) over 3-step simulation (ncvlog, xmelab, xmsim)?

Single step simulation is faster than 3-step simulation because of the following reasons:

- `xrun` uses file extension to determine file type and automatically compiles them using the appropriate compilers. After compilation, `xrun` automatically invokes `xmelab` to elaborate the design, and then `xmsim` to simulate.
- No 5x library structure is needed for compilation.
- In 3-step simulation, users need to call different compilers for different file types. So, 3-step simulation takes more time and needs more effort from users.
- `xrun` can also run in a multi-step mode using `xrun -compile`, `xrun -elaborate`, and `xrun -r`. This mode is the most efficient for AMS regression flows where compilation and elaboration are not required for every simulation.

I want to pass a pre-compiled lib but xrun is not reading `cds.lib` from the project directory, what should I do?

Use `-reflib` to pass the pre-compiled library.

FAQ on Migration from SpectreVerilog to AMS Designer

I am using SpectreVerilog and want to migrate to AMS Designer. Which netlister should I use?

We recommend you to use the AMS UNL netlister. The use model is similar to that of SpectreVerilog and you will be able to use your existing PDKs without any modification.

Note: If you are creating a new design configuration view, use the `AMS` template in Hierarchy Editor.

I am using SpectreVerilog and want to migrate to AMS Designer. Will my existing design configuration view work with the UNL netlister?

Yes. All your SpectreVerilog netlister design configuration views will work with the UNL netlister without modification.

Will the Verilog Library files (-v) and the Verilog Library directories (-y) fields that I use with SpectreVerilog netlister work in the same way with the UNL netlister?

Yes.

You can use these fields to pass Verilog library files and directories using the `-v` and `-y` options to `xrun`. These fields are available on the Main tab page of the AMS Options form (*Simulation – Options – AMS Simulator*).

FAQ on Migration from the Cellview-based Netlister and OSS-based Netlister to the AMS UNL Netlisting Flow

I am a user of the Cellview-based netlister. Can I use the UNL netlister?

Yes, if your original PDK has `spectre` CDF simulation information.

Is there any difference in the netlist created by the Cellview-based netlister and the UNL netlister?

No. There is no syntactical difference between the netlists created by the Cellview-based netlister and the UNL netlister. It is always structural Verilog-AMS structural netlists.

The Cellview-based netlister creates a netlist file named `verilog.vams` for each module in the cellview's 5x library structure. However, the UNL netlister creates only a single netlist file named `netlist.vams` in the netlist directory.

Given that we have a stable Cellview-based netlister, why do we need the UNL netlister?

I am a user of the Cellview-based netlister. Why should I use the UNL netlister?

What are the differences between the UNL netlister and the Cellview-based netlister?

There are four main differences between the Cellview-based netlister and the UNL netlister.

1. The Cellview-based netlister reads the `ams` CDF simulation information from your PDK, whereas the UNL netlister reads the `spectre` CDF simulation information from your PDK. In addition, the UNL netlister uses `spectre` custom netlist procedures in your PDK.
2. The UNL netlister supports the same single step `xrun` executable utilized by the digital design teams so as to keep continuity between digital and analog verification teams, whereas the Cellview-based netlister provides for 3-step simulation using `ncvlog`, `xmelab`, `xmsim`.
3. The UNL netlister supports external text referencing using the black-box design unit (BDU) methodology. External text source code referencing can occur without importing into the 5x DFII environment. This aligns with the digital flow use model.
4. The UNL netlister creates a single `netlist.vams` file whereas the Cell-based netlister creates individual cellview-specific netlists.

What are the advantages of the Cellview-based netlister over the UNL netlister?

The Cellview-based netlister can create a cell-specific netlist automatically when you check-and-save the schematic cellview. The UNL netlister cannot.

What are the advantages of the UNL netlister over the Cellview-based netlister

1. The UNL netlister supports faster simulation using the single step xrun and does not need any 5x library structure, whereas the Cellview-based netlister provides for the 3-step simulation and needs implicit tmp libraries and 5x library structure.
2. The UNL netlister can netlist parametric cells (Pcells) in your design natively, but the Cellview-based netlister cannot without modifying Spectre custom netlisting procedures.
3. The UNL netlister creates a single netlist of all the schematics. This makes it easier for analog designers to hand over the single netlist to verification engineers. Further, as the UNL netlister creates a single netlist and a self-contained `INCA_libs` worklib library with compiled object pak files, it takes less time to elaborate the configuration and is portable. This is unlike the Cellview-based netlister where `xmelab` has to search and read the `cds.lib` files (which might be scattered over your network) to resolve all design units.
4. The UNL netlister utilizes the new HED `pc.db` file implicit Cache temp area to update `pc.db` files for HED 5x. This mimics the hnl directory implicit tmp area in Cell-based netlister for `pc.db` updates.
5. Debugging is easier because the UNL netlister creates a single netlist file.

What changes do I need to make to migrate from the Cellview-based netlister to the UNL netlister?

1. .The Cellview-based netlister does not require the Virtuoso (shadow `netlist.oa` file) database for text cellviews nor does the UNL netlister. Both are shadow-free and symbol-free text-based solutions. Both flows treat text natively.
2. If your design has an instance bound to symbol view and symbol view is in HED stoplist, the UNL netlister treats the instance as a digital external text reference, black-box design unit.

   The UNL netlister was created for ease of migration from all previous AMS netlisting flows. Hence, the UNL netlister has inherited the similar use-models from all of the legacy AMS netlisting flows to ease migration efforts.

There are two ways to make the UNL netlister treat instances bound to symbol view as analog. One way is to edit the design configuration view and bind such instances to `spectre` view. The other way is to add `symbol` in the *Netlist using spectre CDF simInfo* field on the Netlister tab page of the AMS Options form (*Simulation – Options – AMS Simulator*).

New users of the UNL netlister are expected to bind analog instances to the `spectre` view.

I have been using the Cellview-based netlister. Will my existing design configuration view work with the UNL netlister?

Yes.

Your Cellview-based netlister design configuration view will work with the UNL netlister without any modification.

Note: If your design has analog primitive instances bound to `symbol` view, you must change `symbol` view in the Hierarchy Editor stop view list to `spectre`, or add `symbol` in the *Netlist using spectre CDF simInfo* field on the Netlister tab page of the AMS Options form (*Simulation – Options – AMS Simulator*).

I am unable to find the amsControl.tcl file in the netlist directory. Where is the probe Tcl file?

The probe Tcl file is now named `probe.tcl` and is located in the netlist directory.

How does the UNL netlister compile Verilog-A files?

The UNL netlister flow utilizes support for Verilog-A buses, complex port connectivity, and instance and occurrence-based cellview bindings through a new xrun feature enabling auto generation of Verilog-AMS-wrappers around the `ahdl_include` veriloga source.

The Verilog-A files are spectre parsed through `ahdl_include` for better performance (ahdlCMI compiled Verilog-A) and consistency with Spectre flow.

Note: The `ahdl_include` statements specify the path of the Verilog-A text file. This behavior is same across the Spectre and SpectreVerilog netlisters.

As the Verilog-A files are compiled by the analog solver, SimVision or NC debug cannot take place as it can for Verilog-AMS source code.

Note: Even though AMS UNL netlister is a shadow-free text solution, Verilog-A is an exception. Verilog-A still requires a shadow and must be created and edited using the traditional DFII text import method with CV2CV.

How does the Cellview-based netlister compile Verilog-A files?

The Cellview-based netlister compiles Verilog-A files as Verilog-AMS using the `ncvlog -ams` command. Hence it is NC-compiled and you can use SimVision to debug the files.

Why can't I use SimVision for debugging Verilog-A files when I use the UNL netlister?

The UNL netlister compiles Verilog-A files using the analog solver (`ahdl_include` with Spectre or APS). This gives you improved performance and consistency with Spectre.

However, the disadvantage is that you cannot use SimVision to debug Verilog-A files. If you need to debug Verilog-A, a workaround is to rename the `veriloga.va` files to `verilog.vams`. `xrun` will then compile the file using `ncvlog` and you will be able to debug it in SimVision. This is essentially what happens when you use the Cellview-based netlister.

UNL Netlister FAQ

How can I use Verilog text files?

There are five ways to use Verilog text with the UNL netlister.

1. Import the Verilog text file using Verilog In
2. Open and save the Verilog text file in a text editor using CV2CV text import from DFII. This will import the Verilog text into the 5x library (library:cell:view) structure. The path of the Verilog text file is printed in the `textInputs` file (located in the netlist directory) and passed to `xrun` using the `-f` option.
3. Another way, typically used by SpectreVerilog and UltraSimVerilog users, is to use the symbol view as an HED stopping view and pass the Verilog text file separately using the `-v` or `-y` option. You can use the *Verilog Library files (-v)* and the *Verilog Library directories (-y)* fields in the Main tab page of the AMS Options form (*Simulation – Options – AMS Simulator*) for this purpose.

4. Use the new HED Set Cell View binding property called *Mark As External HDL Text*. This mimics the same symbol stopping view behavior intended to blackbox a cellview to HED and netlisting visibility pointing to external text outside of the DFII environment.
5. The most common import step for text within the AMS flow is to perform a command line `ncvlog -use5x` compilation of the text source code. This results in creating the HED 5x library.cell:view structure so as to register the text for use in HED view switching.

How can I use VHDL text files?

If your design configuration uses VHDL text files, both the entity view and the architecture view should exist. If you do not have the entity and architecture views already in DFII, use traditional DFII text import (CV2CV) to import the VHDL text file. VHDL import will create an entity view and an architecture view. The name of the architecture view will be the same as the name of the architecture defined in the VHDL file. The `entity/vhdl.vhd` file will have the entity definition, while the `<`*`architecture`*`>/vhdl.vhd` file will have the definition of that particular architecture.

For the design configuration view in the Hierarchy Editor (HED), you should bind the instance to the architecture view, and not to the entity view. You must also add the created architecture names to the HED view-list as valid architecture views to bind to.

Both the entity view `vhdl.vhd` and the architecture view `vhdl.vhd` files are sent to `xrun` (the filepaths are printed in the `textInputs` file present in the netlist directory).

Can I use VHDL-AMS text with the UNL netlister?

Yes.

Can the UNL netlister create a VHDL-AMS netlist?

No.

The netlister creates a Verilog-AMS netlist only.

How does the UNL netlister pass the HDL text files to xrun?

The UNL netlister traverses the design and obtains a list of cells that have the following DFII imported view-types:

- Verilog (Verilog text)

- VHDLAMSText (VHDL-AMS text)
- systemVerilogText (SystemVerilog text)
- VerilogAMSText (Verilog-AMS text)
- vhdl (VHDL text)
- Veriloga (Verilog-A text)

These are considered white-box design units (WDU) due to their visibility in HED and accessibility to cellview switching

The HDL filenames are then printed in the `textInputs` file which is located in the netlist directory. The `textInputs` file is passed to `xrun` using the `-f` option.

Note: The configuration's collection of text source files used in the design is defined in the textInputs file. UNL's advanced AMS xrun binding capabilities allow for true cell, instance, occurrence, and library-based bindings for text as defined in this file.

Are the values of -incdir, -v, -y and -f options relative to the current working directory, or relative to netlist directory?

The value that you specify for the `-incdir`, `-v`, `-y`, and `-f` options are relative to the current working directory (CWD) from where you started the Virtuoso workbench.

Where are the model files searched?

Model files are searched in the directories specified in the *Include Paths* field in the Simulation Files Setup form (*Setup – Simulation Files*).

You can specify individual model files in the Model Library Setup form (*Setup – Model Libraries*) or specify the directory that contains model files in the *Include Path* field in the Simulation Files Setup form (*Setup – Simulation Files*).

Note: If you are specifying the relative path to a model file in the Model Library Setup form, ensure that the path is relative to the current working directory (the directory from which you started the Virtuoso workbench).

The paths to model files specified in the Model Library Setup form are passed to `xrun` using the `spiceModels.scs` file, while the value of the *Include Path* field is passed using the `-modelincdir` option.

Note: The `-modelpath` option has been deprecated and is replaced by another mechanism called `amsd_subckt_bind=yes` settings which reside in the `spiceModels.scs` file.

I chose the Spectre solver, but I see both Spectre control file (amsControlSpectre.scs) and UltraSim control file (amsControlUltraSim.scs) in the netlist directory. Why?

Although you chose either the Spectre or the UltraSim analog solver, ADE Explorer create the AMS control files for both the analog solvers. This is done with the assumption that you might later change the solver, and as the control files for both the solvers are available, netlisting and simulation will take lesser time.

However, the AMS control file of only the analog solver that you chose is passed to `xrun`. If you read the `runSimulation` file in the netlist directory, you will see the correct AMS control file specified against the `-analogControl` option.

My design has two instances of the same library and cell, one bound to verilog view, and the other to Verilog-A view. Will the UNL netlister netlist and bind the instances correctly?

My design has two instances of the same cell, one bound to verilog view, and the other bound to verilogams view. Will UNL netlister netlist the instances correctly?

Yes.

The UNL netlister uses the powerful AMS xrun binding engine (BIND2) specifically created to perform all types of cellview bindings possible within a config. Full support exists for cell, instance, occurrence, library, same cell from different libraries. All combinations of text and schematic are supported in any hierarchical configuration.

My design has two instances of same cell but they are bound to different views (schematic and verilog). Will the UNL netlister netlist the instances correctly?

Yes.

I see xmelab CUVMUR errors; model name in instance definition is incorrect. What is the solution?

`CUVMUR` is an error mnemonic from the `ncvlog` and `xmvhdl` compilers. It means that the elaborator could not resolve or find the cell or module to elaborate. In other words, the module was not compiled, therefore was unresolved for elaboration. Check the `xrun.log` for proper compilation of the module before elaboration.

For example, if you see the following `xmelab` error in `xrun.log`:

```
vpulse #(0.05, 0.05)   V0 ( .PLUS(net019), .MINUS(cds_globals.\gnd! ));
xmelab: *E,CUVMUR (./netlist.vams,133|24): instance 'verilogtesttop.V0' of design
unit 'vpulse' is unresolved in 'worklib.verilogtesttop:vams'.
```

The instance definition of `V0` in netlist is:

```
vpulse #(0.05, 0.05)   V0 ( .PLUS(net019), .MINUS(cds_globals.\gnd! ));
```

In the schematic, `V0` is an instance of `vpulse`. Hence, instead of model name, the cell name has been printed.

Further, the parameter name-value pairs in the instance definition are not printed either.

A common reason for this problem is that you are using a configuration that you used for netlisting using the Cellview-based netlister. The UNL netlister treats instances bound to symbol view with symbol view in HED stoplist as digital stopping views. Consequently, the netlister does not read the `spectre` CDF simulation information, prints incorrect model name, and does not print the parameter name-value pairs in the instance definition.

While netlisting, you would have got the following pop-up window with the following message:

```
The following cells will be netlisted without using the spectre CDF simulation
information, because they are bound to view symbol:
< list of cells>
To use the spectre CDF simulation information, do one of the following
Open your configuration in Hierarchy Editor and replace "symbol" with "spectre" in
the view list and stop list.
Choose Simulation->Options->Netlister and add "symbol" in the "Netlist using
spectre CDF simInfo" field.
```

As specified in the message, the solution is to either edit the configuration, or add `symbol` in the *Netlist using spectre CDF simInfo* field on the Netlister tab page of the AMS Options form (*Simulation – Options – AMS Simulator*) to have symbol view act as an analog stopping view.

Renetlist the design, and check that the netlist has the correct model name and all the parameter name-value pairs.

Why do I get SFE-23 errors during UNL Netlisting Step?

```
Running netlist assembly..

   ERROR (SFE-23): "analog/input.scs" 36: The instance `I153|NM0' is
referencing an undefined model or subcircuit, `nmos25
```

The error refers to the portion of the UNL netlisting step which calls Spectre to parse the model cards could not resolve the definition of the analog primitive model nmos25.

UNL Netlisting requires that the model libraries for analog primitives be setup prior to netlisting. The path of the model file will be either missing or incorrect. Open the Model Library Setup form (*Setup - Model Libraries*) and correct the value.

Same setup requirement applies for Design Variables. These must be defined prior to UNL netlisting.

What does the Clean snapshot and pak files check box in the Netlist and Run Options form do? When should I use it?

The *Clean snapshot and pak files* check box appears in the Netlist and Run Options form (*Simulation – Netlist and Run Options*) when you select *AMS Unified netlister with xrun*.

By default, this check box is deselected. If you select this check box, the `INCA_libs` directory inside the netlist directory is deleted before calling `xrun`. The `INCA_libs` directory contains the simulation snapshot and the `.pak` files. Hence, if you choose to delete the `INCA_libs` directory, it essentially will lead to re-compilation, and hence re-elaboration and re-simulation of the design.

If you select the *Clean snapshot and pak files* check box, the `-clean` option is passed to `xrun`.

How does the UNL netlister print inherited connections in the netlists?

The UNL netlister prints the inherited connections (net expression wire labels and netSet properties) using the `cds_net_set` compiler attribute.

Consider a module that has a wire segment with the following net expression label: `@n3vdd:%:vdd!`

For this net expression, the netlist will have the following definition:

```
wire (*
integer inh_conn_prop_name = "n3vdd";
integer inh_conn_def_value = "cds_globals.\\vdd! ";*)
cdsNet0;
```

Somewhere in the design hierarchy, a `netSet` property is defined on an instance. The `netSet` property appears as follows in the instance definition:

```
<modelName> (* integer cds_net_set
[0:2] = { "n3vdd","n2vdd","n2gnd" };
integer n3vdd = "cds_globals.\\s_vdd! ";
*) <instName>…
```

Where can I find basic information on how to use inherited connections with Cadence tools?

A tutorial on how to use inherited connections with various Cadence tools in the design flow is available at `<your installation directory>/tools/dfII/samples/tutorials/inhconn`. You can refer to the *Inherited Connections Flow Guide* on the Cadence Online Support website to run through this tutorial.

To understand how to set net expression wire labels and netSet properties in Virtuoso Schematic Editor, read the *Inherited Connections* section in the *Virtuoso Schematic Editor User L Guide*.

Can the UNL netlister print inherited connections as pseudo ports, like the Spectre and SpectreVerilog netlisters?

No.

The UNL netlister uses the `cds_net_set` compiler attribute to print the inherited connections.

Do sideways inhconn netSets work with the UNL netlister?

Yes.

As the UNL netlister uses compiler attributes to print netSet and net expressions in the netlist, sideways netSet works. This is particularly useful when your design uses inherited connection connect modules with built-in net expression attributes used for multiple supply designs.

Miscellaneous FAQ on AMS Designer Interface in ADE and ADE Explorer

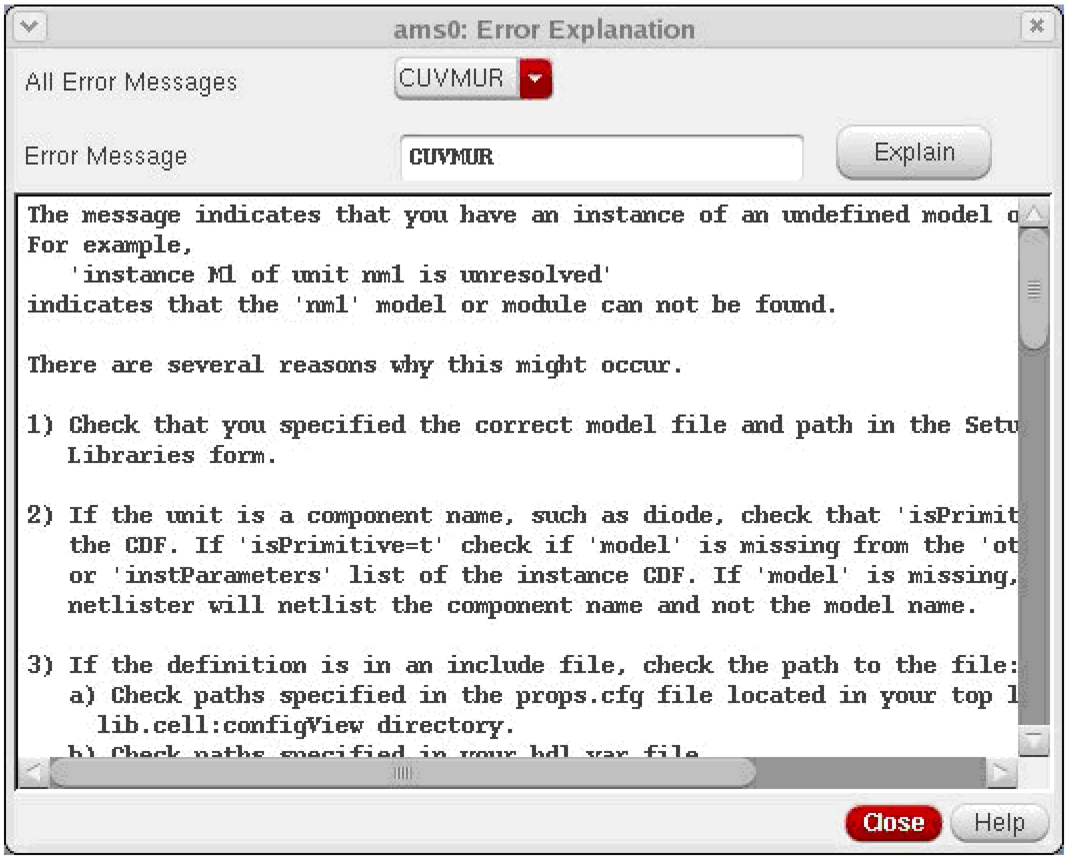
I am adopting AMS Designer in ADE Explorer. Which netlister should I use?

We recommend you to use the AMS UNL netlister. The Cell-based and AMS OSSN netlisters will be eventually deprecated once all users migrate to AMS UNL.

How can I get more information on an error from the simulator?

Select *Simulation – Output Log – Error Explanation*. You will see a list of error mnemonics in the *All Error Messages* pull-down menu. These are the error mnemonics reported by `xrun` when you simulated your design. The software reads all error mnemonics in the `xrun.log` file and populates the pull-down menu with those mnemonics.

Select the mnemonic for which you want to read the explanation, and click *Explain*.



```
The message indicates that you have an instance of an undefined model o
For example,
   'instance M1 of unit nm1 is unresolved'
indicates that the 'nm1' model or module can not be found.

There are several reasons why this might occur.

1) Check that you specified the correct model file and path in the Setu
   Libraries form.

2) If the unit is a component name, such as diode, check that 'isPrimit
   the CDF. If 'isPrimitive=t' check if 'model' is missing from the 'ot
   or 'instParameters' list of the instance CDF. If 'model' is missing,
   netlister will netlist the component name and not the model name.

3) If the definition is in an include file, check the path to the file:
   a) Check paths specified in the props.cfg file located in your top l
     lib.cell:configView directory.
   b) Check paths specified in your hdl.var file
```

You can also type any error mnemonic (not necessarily reported by `xrun` in the last simulation) in the *Error Message* field and click *Explain* to view the explanation.

You can also find the error mnemonic from nchelp command line by doing the following:

```
nchelp xmelab ERROR_MNEMONIC
```

I defined a new value for a variable in the CIW and clicked on the Netlist and Run icon. However, I still see the same incorrect netlist. What is wrong?

If you have defined a variable, you need to recreate the netlist, that is, choose *Netlist – Recreate*. Then you can either choose *Simulation - Netlist and run* or click the *Netlist and Run* button.

I do not have xrun in my older IUS release, but I have the ncverilog executable. Can I use ncverilog with the UNL netlister?

No.

The UNL netlister supports simulation with only `xrun`. If your IUS release does not have the `xrun` executable, you will see the following error message:

```
Failed to create the netlist because either the xrun executable is not in a
directory listed in the 'path' variable or you are running IUS 8.X in an unsupported
platform.

Correct the value of the 'path' variable or move to a supported platform and then
try again.
```