CDL Out Task Assistant

Product Version IC23.1
June 2023

Contents

1

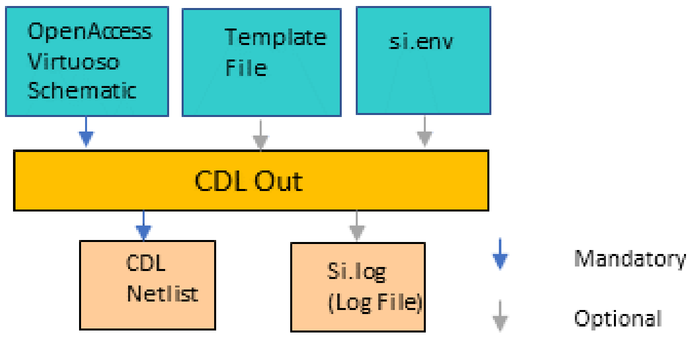
What is CDL Netlisting?

Circuit Description Language (CDL) is an OSS based netlister and is a subset of the Simulation Program Integrated Circuit Emphasis (SPICE) language. Netlist representation of the schematic design is needed to compare a layout versus a schematic (LVS). CDL is one of the formats used for LVS comparison.

How to Create CDL Netlists?

You can create CDL netlists using **CDL Out** or the **Analog and Microwave Circuit Description Language (auCDL)**

CDL Out translates an OpenAccess Virtuoso-Schematic design into a CDL netlist, which is suitable for verification products like Assura, Diva, and Dracula. CDL Out is useful in creating CDL netlists of digital designs. The following figure shows the inputs and outputs of CDL Out.



You must specify an `si.env` file when you use the `si` command to run CDL Out. The `si` command is the noninteractive, batch mode version of CDL Out.

Analog and Microwave Circuit Description Language (auCDL) is the netlister that is used to create CDL netlist for analog designs.

How to Create CDL Netlist for Digital Designs Using CDL Out?

How to Create CDL Netlists for Analog Designs?

CDL Out Form Help

Which Licenses are Needed for CDL Netlisting?

You must have one of the following licenses to run auCdl from Virtuoso or from the command-line. If one of these licenses are not already checked out, the first available license is checked out in the following order when you run auCdl:

- 95100 Virtuoso® Schematic Editor L
- 95115 Virtuoso® Schematic Editor XL
- 206 Virtuoso® Simulation Environment

Which Files are Used in CDL Netlisting?

What is the .simrc file used for?

What is the si.env file used for?

What is the .simrc file used for?

You can use the simulation run control (`.simrc`) file to customize simulations. This file lets you set your defaults for the simulation variables. It overrides the contents of the `si.env` file. Therefore, the `.simrc` file provides a mechanism to set the defaults at the user or flow level. This file is optional and is loaded if it exists.

If you want to use another file, instead of `.simrc`, to customize simulations for a specific flow, use the UNIX environment variable `ossUserSimrc` to set the path of that custom file. For example, the following command sets `mysimrc` as the custom file.

```
$ setenv ossUserSimrc mysimrc
```

The functionality provided by `ossUserSimrc` lets you customize the simulation environment to suit the specific requirements of different flows, without changing your `.simrc` file. If `ossUserSimrc` is not set, then the default CSF mechanism is used to load the simulation customization file.

How to Customize Netlists Using the .simrc File?

What is the si.env file used for?

The simulation environment (`si.env`) file is read in from the simulation run directory. It can be created manually or by using the CDL Out form in the run directory. This file is used to instruct simulation environment (SE) which design to simulate and which simulator to use. In addition to the default information SE stores in this file, you can instruct SE to save any application-specific variables by using the variable `simSimulatorSaveVars`. The file is overwritten by the SE function `simPrintEnvironment` each time a simulation is run, and any other information stored in this file that SE has not been informed of is lost. The `si.env` file is used to communicate environment information between SE and the Cadence graphics program.

The `.simrc` control file, which can be searched using the CSF mechanism, always overwrites the values in the `si.env` file.

What is the .simrc file used for?

Sample si.env File

```
simLibName = "opus"
simCellName = "latch.cdl"
simViewName = "schematic"
hnlNetlistFileName = "netlist"
simRunDir = "/cds/1.0/test/translator/cdlout/paramCase/"
simSimulator = "cdl"
simViewList = '("cdl" "schematic" "gate.sch" "symbol")
simPrintInhConnAttributes = 'nil
simNetNamePrefix          = N
simInstNamePrefix         = X
simModelNamePrefix        = M
hnlMaxLineLength          = 79

preserveALL  = t
preserveRES  = t [XX]
shortRES     = 250.000000
checkRESSIZE = t
preserveCAP  = t
checkCAPAREA = t
preserveDIO  = t
checkDIOAREA = t
retainBusses = t
CDLUsePortOrderForPinList = 'nil
```

For information on `sin.env` properties and variables, see:

Description of si.env Properties

Variables in the si.env File

2

How to Create CDL Netlists for Analog Designs?

The following topics discuss how to create and customize auCdl netlists for analog designs:

What Do I Need to Run auCdl?

You can run auCdl from within or outside the Virtuoso (DFII) environment.

To translate files from the DFII database format into an auCdl netlist,

1. Set the CDS_Netlisting_Mode environment variable as given below:
   `setenv CDS_Netlisting_Mode "Analog"`
2. Create an auCdl view for the stopping cell.
3. Define `netlist procedure` in Stopping cell CDF.

You can customize the auCdl Netlister using the simulation run control (`.simrc`) file.

How to Define ‘netlist procedure’ in Stopping Cell CDF?

How to Define Netlist Procedure in the Stopping Cell CDF?

1. From CIW, choose *Tools – CDF – Edit* to open the Edit Component CDF form.
2. Select *Scope* as *Cell*.
3. Select *CDF Layer* as *Base.*
4. Select *Library Name*.
5. Select *Cell Name*.
6. Select *Simulation information* tab and choose *auCDL* as simulator.
7. Define or edit the appropriate *Netlist Procedure* name.
8. Press *OK*.

How to Create auCdl Netlists using Virtuoso?

In DFII, you can extract the auCdl netlist by doing the following:

1. Select *File – Export – CDL*.
   The CDL Out form appears.
2. In the *Top Cell Name* field, type the name of the top cell.
3. In the *Library Name* field, type the name of the OpenAccess library containing the view you want to netlist.
4. Select *Netlisting Mode* as *Analog*.
5. Specify values for the optional fields you want.
6. Click *Apply* or *OK*.
   CDL Out begins the translation.

Design Translation Using CDL Out

How to Run auCdl from Command-line?

To run auCdl from the command-line, you must create a simulation environment (`si.env`) file in advance and name the file as a command argument. Run CDL Out interactively once to create the `si.env` file. After the `si.env` file is created,

1. Copy the `cds.lib` file to the run directory.

2. Enter the following command:
   ```
   si -batch -command netlist
   ```
3. The -batch option runs CDL in batch mode and the -command netlist option generates an ASCII netlist file.
4. CDL Out can generate a hierarchical netlist. CDL Out generates a netlist hierarchy that duplicates the hierarchy of your design. Each cell in your schematic becomes a separate subcircuit in the netlist. The hierarchical netlister automatically prefixes each instance name with the proper character for its element type; for example, `"M"` for MOSFET and `"R"` for resistor. This prefixing minimizes mapping and name translation.

What is the si.env file used for?

How to a Create config View for auCdl?

To create a config view for auCdl,

1. In CIW, choose *File - New - Cellview*.
   The Create New File form appears.
2. In the *Cell Name* field, enter the name of the cell in which you want to create the config view.
3. In the *View Name* field, enter the name of the view you want to create—for example, config_aucdl.
4. In the *Tool* cyclic field, select *Hierarchy - Editor*.
5. Click *OK*.
6. Specify the top-level cell name and its view.
7. Click *Use Template*.
   The Use Template form appears.
8. In the Name cyclic field, choose auCdl.
9. Click *OK* to display the New Configuration form.
   You can modify the view list.
10. Click *OK* to create the view.
11. Choose *File – Save* to save the configuration.

How to Include Partial Netlist File in SUBCKT Calls?

You can automatically bind your cells to source files which will then be included in the .subckt statements.

Add following in your `.simrc` file

```
hnlReadHdbProps = 't
```

```
ansCdlHdbFilePathProp = "<property name>"
```

Using Hierarchy Editor, add the property to the lib/cell/view as cell property in which the netlist needs to be included. In the value field of this property, define full path of the partial netlist file and netlist the config view of the top cell. After the netlist is complete the information is added to the `subckt` file.

For more information, see Including partial netlist file in SUBCKT calls.

3

How to Create CDL Netlist for Digital Designs Using CDL Out?

CDL netlisting for digital designs are created using the CDL Out form. The following topics discuss how you can use the CDL form for digital netlisting.

How to Prepare the Virtuoso-Schematic for CDL Out?

The lowest levels of the schematic must contain primitives that CDL Out recognizes; for example, transistors, resistors, and capacitors. You must extract and save all levels of the schematic hierarchy before you netlist your design using CDL Out.

If you work on a circuit-level schematic and you create a circuit cell, which uses different parameter values for different instances of that cell, parameter inheritance is important. For example, you might generate an inverter in your library and use it with 1X drive strength in one cell and with 2X drive strength in another. To ensure that the correct parameter values are passed to the appropriate instances, pass them through the hierarchy.

When you use hierarchical netlisting and parameter inheritance at the same time, special limitations apply. For information about these limitations, see *How CDL Out Translates Parameters*.

How CDL Out Translates Parameters?

How to Create an Instance Array?

How to Create a Black Box?

How to Run CDL Out from GUI?

How to Create a Black Box?

Using black box properties, you can direct CDL Out to not netlist various blocks in a design. You add the black box properties directly to the property list of the symbol view.

The black box feature blocks out a particular cell by adding properties from the symbol view.

You can add the following properties to the symbol view of the instance master:

Name	Property Type
`CDLBlackBox`	`t / nil`
`CDLGenerateSubcircuitCard`	`t / nil`
`CDLPinList`	`string`
`CDLParameterNameList`	`string`
`CDLParameterValueList`	`string`

Using the Black Box Properties

How to Create an Instance Array?

CDL Out lets you make an array from an instance. The specific instance is expanded with the name and property type. You can choose the number of times you want CDL Out to expand the instance.

Name	Property Type
`CDLMultiplier`	integer

For example, if CDLMultiplier = 3 on instance MP8, the netlist appears as follows:

```
MN8.1          VDD        IN       OUT      C21PLUS            P
```

```
MN8.2          VDD        IN        OUT        C21PLUS              P
MN8.3          VDD        IN        OUT        C21PLUS              P
```

CDL Out Form Help

How to Run CDL Out from GUI?

To run CDL Out, perform the following steps by using the CDL Out form:

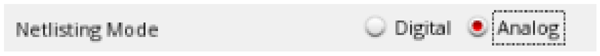
1. Select *File – Export – CDL*.
   The CDL Out form appears.
2. In the *Top Cell Name* field, type the name of the top cell.
3. In the *Library Name* field, type the name of the OpenAccess library containing the view you want to netlist.
4. Select *Netlisting Mode* as *Digital*.
5. Specify values for the optional fields you want.
6. Click *Apply* or *OK*.
   CDL Out begins the translation.

⚠ When you click *OK*, all settings you made in the CDL Out form are saved into the `si.env` file, which is created inside the run directory. The information in the `si.env` file is overwritten every time you change the field value. However, if a `.simrc` file exists, then `.simrc` overrides the current `si.env` values from the CDL Out form.

CDL Out Form Help

How to Set the Netlisting Mode?

By default, CDL Out supports two netlisting modes, *Digital* and *Analog*. The *Digital* mode is selected by default if CDS_Netlisting_Mode environment variable is not set. If the `CDS_Netlisting_Mode` environment variable is set, the value of `CDS_Netlisting_Mode` is considered.


Netlisting Mode Digital [✓ Analog]

Valid Values:

Digital	Selects CDL Out.

Analog	Selects auCdl.

Default value: ***Analog***

For creating a digital netlist, select the *Digital* option.

How to Run CDL Out from Command-Line?

1. Prepare the `si.env` file. Among other information, the `si.env` file carries the name of the design to be netlisted.

   For more information, see How to Prepare si.env File for CDL Out?.

2. Type the following at the command line:

   ```
   si -batch -command netlist
   ```

How to Set Up Automatic Netlist Inclusions?

This feature allows you to include the CDL netlist of the blocks, which do not have a schematic view, in the final top level CDL netlist. During CDL Out, it is helpful if some of the blocks in the design do not have a schematic view but their CDL netlist is available. For such blocks, perform the following steps:

1. Create a blank schematic view.
2. In the schematic view, create dummy pins. The pin names should match with the pins present in the corresponding symbol view.
3. Create the string property `nlAction` on the schematic and symbol view and set its value to stop.
4. Create a boolean property `CDL_INCLUDE_FILE` on the schematic view and set its value to `TRUE`.
5. Copy the CDL netlist file `cdlNetlist` of this cell in the schematic view directory.

Now, if you run CDL out on the top level cell, then it will not create the `.subckt` in the netlist for the cellviews which have property `nlAction` set as stop. For such cellviews, if the `CDL_INCLUDE_FILE` property is set to true, then its netlist file `cdlNetlist` is copied as:

```
./includeFiles/libName_cellName_viewName
```

All these files are included at the bottom of the netlist with the help of `INCLUDE` statement.

How to Prepare a Template File for CDL Out?

A template file is a collection of file names and option values that you can load into the CDL Out form. You can create a CDL Out template file in one of two ways:

- Enter values in the CDL Out form and click the *Save* command to save the option values to the file you specify as the template file.
- Create a text file. You can copy the sample template file `cdlOut.il` in the samples/transUI directory and modify the file.

The required values in the CDL Out template file are the name of the OpenAccess library containing the top level cellview and the top cell name to translate. The other values are optional.

Sample Template File

The following is the sample CDL Out template file Cadence supplies in `samples/transUI/CDLOut.il`:

```
cdlOutKeys = list(nil
  'simLibName           "opus"
  'simCellName          "latch.cdl"
  'simViewName          "schematic"
  'hnlNetlistFileName   "netlist"
  'simRunDir            "."
  'shortRES             2000.000000
  'resistorCheck        "value"
  'capacitorCheck       "value"
  'diodeCheck           "both"
  'displayPININFO       t

  cdlOutKeys = list(nil
  'simLibName           "opus"
  'simCellName          "latch.cdl"
  'simViewName          "schematic"
  'hnlNetlistFileName   "netlist"
  'simRunDir            "."
  'shortRES             2000.000000
  'resistorCheck        "value"
  'capacitorCheck       "value"
```

```
'diodeCheck             "both"
'displayPININFO         t
```

How to Prepare si.env File for CDL Out?

To run CDL Out from command line, you must prepare an `si.env` file. The `si.env` file contains few mandatory and few optional variables. If you run CDL Out from the CIW, CDL Out creates an `si.env` file automatically in the run directory.

In case, the netlister is invoked from the command line by using:

```
si -batch -command netlist command
```

and `checkScale` is set to nil in `si.env` then CDL Out does not print `*.SCALE` statement in the netlist.

What is the si.env file used for?

What is the .simrc file used for?

Variables in the si.env File

4

How CDL Out Translates Data?

There are important differences between CDL format and the OpenAccess Virtuoso-Schematic view. CDL Out generates a netlist hierarchy that duplicates the hierarchy of your design. Each cell in the schematic becomes a separate subcircuit in the netlist. The hierarchical netlister automatically prefixes each instance name with the proper character for its element type; for example, `"M"` for MOSFET and `"R"` for resistor. This minimizes mapping and name translation.

CDL Out names instances and nets differently than a flat netlister. A flat netlister maps all the names to unique names. This avoids naming conflicts if you use identical names for instances in different cells of your schematic.

How Illegal Names are Mapped during CDL Out?

Unlike a flat netlister, CDL Out maps only illegal names to new names in the hierarchical netlist. CDL Out considers a name illegal if it contains illegal characters or is more than 60 characters.

How Illegal Characters are Mapped during CDL Out?

CDL Out searches first for illegal characters, which it maps, as shown in the following table.

First Character	Net	Instance
`$ ; : = \t <space>`	`N`	`X`
`0 to 9`	`N0 to N9`	`X0 to X9`
`A to Z`		`XA to XZ`
`a to z`		`X_A to X_Z`

For example, if `$333` is a net name, CDL Out maps it to `N333`. If `$333` is an instance name, CDL Out maps it to `X333`.

To avoid conflicting instance names, if a name uses less-than and greater-than signs (`<` and `>`), CDL Out strips the greater-than sign from the name and replaces the less-than sign with an underscore. For example, the instance named `X1<0>` would be mapped to `X1_0`.

How Longer Names are Mapped during CDL Out?

If a name has more than 60 characters, CDL Out maps it to a unique number preceded by one of the following letters:

N	for net names
I	for instance names
M	for macro names

Which Middle Illegal Characters are Removed during CDL Out?

CDL Out strips the following middle illegal characters from the instance names and net names:

semicolon	`;`
colon	`:`

equals sign	`=`
dollar sign	`$`
<space>	`" "`
tab	`\t`

⚠ If *Map Bus Name From <> To []* check box is selected in the CDL Out form then CDL Out strips middle illegal characters from net names.

CDL Out strips the following middle illegal characters from the net names:

period	.
open parentheses	(
close parentheses	)
open bracket	[
close bracket	]
dollar sign	$
exclamation point	!
plus sign	+

How CDL Out Translates Parameters?

Circuit-level simulations frequently involve element and model parameters. When combining parameter inheritance and hierarchical netlisting, Design Framework II parameter inheritance mechanisms are more general than those that CDL supports. The graphic display of the schematic might show different parameter values than those printed in the netlist.

To avoid this problem, when you create your schematics follow the guidelines for:

- Fixed-Value Inheritance
- Assigned Parameters

- Parameter-Value Inheritance
- Inheritance of Inherited Parameters

For information on these parameters, see:

Guidelines for Parameters in CDL Out

How are Inherited Connections Supported in CDL Out?

When CDL Out netlists a design that contains inherited connections, it creates dummy ports (or pseudo ports) to maintain connectivity across modules and their instantiations. These dummy ports introduce unwanted nets in the hierarchy and also result in the loss of data when design information is shared across various EDA tools in the flow.

The switch is a SKILL environment variable named.

You can turn off the creation of the dummy ports by using the CDL Out GUI check box option Print Inherited Connections. The corresponding switch in the si.env file is 'simPrintInhConnAttributes'. It is a boolean variable with the default value 'nil'. This means that in the default state CDL Out will continue to create the dummy ports for inherited connections in a design. You can turn off the creation of the dummy ports by setting this variable 't'. You can set this environment variable in the .simrc file.

For more information, see:

Enhanced Support for Inherited Connections Enhanced Support for Inherited Connections

How CDL Out Translates Instances of Primitive Devices?

You need to define an instance of a primitive device in the schematic so that CDL Out can netlist the instance. CDL Out recognizes instances of only those cells that have both a symbol view and a CDL view. You can store default values of parameters in the symbol view so that they are displayed in the schematic. Store the default values in the CDL view so that they are printed in the output CDL netlist.

To create CDL views, use the Symbol/Simulation Library Generator (S/SLG) described in the Virtuoso Schematic Editor User Guide.

For the complete list of formatting functions (`hnlCDLFormatInst`) and parameter list variables (`hnlCDLParamList`) for all the primitive cells supported by CDL Out, see:

List of formatting functions and parameter list variables for primitive cells supported by CDL Out

How CDL Out Translates Global Signals?

CDL Out interprets a signal that ends with a `!` as a global signal. The signal name appears in the `.global` and `.pin` statements. If the global signal is associated with a component, the signal does not appear in the I/O signal list of a subcircuit definition.

Do not use global signals with a hierarchical input connector unless the schematic is the top-level block of the design. If you use the global signal with a hierarchical input connector, the signal appears in the I/O signal list of a subcircuit definition and creates warnings when you run LOGLVS. Assign wire or label connections directly to instances in the schematic (for example, when you use globals with components.)

For nmos, pmos, and cap devices in the Cadence Library, by default a substrate connection to power or ground is made in the netlist. To override the VDD and GND defaults for these connections, include the following declarations in the .simrc file:

```
hnlCDLNMOSBulkNetName="cvss!"

hnlCDLPMOSBulkNetName="cvdd!"

hnlCDLCAPBulkNetName="dvss!"
```

In this example, `cvss` is the bulk connection for the nmos device, `cvdd` is the bulk connection for the pmos device, and `dvss` is the bulk connection for the cap device. The signals appear as `CVSS!`, `CVDD!`, and `DVSS!` in the netlist.

Just like in the schematic where the global signal names are suffixed with `!`, the global signal names in the output CDL netlist file also end with `!`.

Which CDL Formats are Generated for Primitive Components?

The CDL view for primitive cells must contain hnlCDLFormatInst, hnlCDLParamList and hnlCDLElementSubType in its property list. The following the CDL Out formats for the primitive components are generated by CDL Out and the cdslib components that use them.

- BJT Element

- BSIM3SOI Element
- Cap Element
- Capacitor Element
- Diode Element
- Inductor Element
- MOSFET Element
- NMOSFET Element
- NPN Element
- PMOSFET Element
- PNP Element
- Res Element
- Resistor Element
- Transmission Line Element
- Voltage Source Element
- Voltage Controlled Current Source Element
- Voltage Controlled Voltage Source Element
- Current Controlled Current Source Element
- Current Controlled Voltage Source Element
- Current Source Element

For more information on these formats, see:

CDL Out Formats

Which Output Files are Generated During CDL Out?

Depending on what output files you specified in the CDL Out form or template file, CDL Out produces one or more of the following files. CDL Out writes all messages to the si.log file.

- Error messages, preceded by "*Error*", indicate serious, unrecoverable conditions.The converted file is incorrect. For example, if a cell instance is called but its master cell does not exist, CDL Out writes an error message to the `si.log` file.

- Warning messages preceded by "`*Warning*`" indicate unexpected but recoverable conditions. The converted file is usable. For example, if a string is too long and has been truncated, CDL Out writes a warning message to the `si.log` file.
- Information messages, preceded by "`*Info`*", indicate the status of a process that is running or the results of a completed process. Information messages are written to the file si.log.
- Statistical messages that CDL Out generates as it runs are also written to the `si.log` file. You can use these messages to estimate the size of the file based on information such as the number of cells and terminals.

How to Customize CDL Netlist?

CDL Out defines various hnl functions and variables that control the CDL Out netlist format. If these func tions and variables are already defined, CDL Out does not overwrite them. You can override the value of variables or functions used in CDL Out by defining them before invoking CDL Out. This can be done by loading a SKILL file before running CDL Out or by defining the variables in .simrc or .cdsinit file. To see details about the various variables and functions that can be overridden, see Customizing the Hierarchical Netlister (HNL).

Example - Customizing CDL Netlist

The variable hnlCommentStr defines the string that is placed at the beginning of a comment in the output netlist file. By default, '*' is used by CDL Out to represent the comments. If you put the value of hnlCommentStr variable to '>' in the .simrc file, then the comments start with '>'.

For information about CDL Out SKILL functions, see the Design Data Translators SKILL Reference.

5

How to Customize Netlists Using the .simrc File?

The behavior of the netlist can be further controlled using the simulation run control (.simrc) file. The parameters that you can include in the .simrc file are described in this section. The parameters you can set in the .simrc file are the same as those that are defined using the simSetDef SKILL function. This SKILL function defines variables only if they have not been defined previously (that is, during initialization when the si.env and .simrc files are read).

Commonly Used .simrc Parameters

For more customization information, see:

Customizing the .simrc file

Customizing the Hierarchical Netlister (HNL)

How to Use auCdlCDFPinCntrl?

You can use `auCdlCDFPinCntrl` to let the CDF termOrder to dictate pin ordering of the top-level cell or the cell that has the auCdl view. The default is `'nil`.

```
auCdlCDFPinCntrl = 't
```

You can also set this parameter to disable the auCdl-50 warnings by specifying either of the following:

```
auCdlCDFPinCntrl = '("cdfTermOrder" "masterPortOrder" "default")
```

Or

```
auCdlCDFPinCntrl = '("cdfTermOrder" "masterPortOrder" "alphaNum")
```

For more customization information, see:

Specifying the Terminal Order for Terminals

How to Use auCdlDefNetlistProc?

You can use `auCdlDefNetlistProc` to generate netlist having connections by name (explicit) or connection by order (implicit)

For implicit netlist:

```
auCdlDefNetlistProc = "ansCdlSubcktCall"
```

For for explicit netlist:

```
auCdlDefNetlistProc = "ansCdlHnlPrintInst"
```

How to Use auCdlEnableNetlistInclusion?

You can use `auCdlEnableNetlistInclusion` to automatically include cdl netlists of subckt used in the schematic by adding `CDS_NETLIST_FILE` property on dummy auCdl view.

```
auCdlEnableNetlistInclusion = 't
```

For more customization information, see:

Including a ROM-Insert Netlist Automatically Into the auCdl Netlist

How to Use auCdlModuleNameMapFunc?

You use to `auCdlModuleNameMapFunc` to rename cell names in the auCdl netlist. For example, you can rename same cell names that in different libraries.

```
auCdlModuleNameMapFunc = custom_ Function_Name
```

For more customization information, see:

Customizing the .simrc file

How to Use hnlUserShortCVList?

You can use `hnlUserShortCVList` to specify the cellview names of the devices to be removed in a list. Defined for devices that have only two terminals. Cadence recommends that you use `hnlUserMultiTermShortCVList` instead of `hnlUserShortCVList`.

```
hnlUserShortCVList = list(
   ;all cells from this library
    "libN"
   ;cell1, cell2 and cell3 from lib1
    list("lib1" "cell1" "cell2" "cell3")
   ;all cells from this library
   list("libM"))
```

For more customization information, see:

Customizing the .simrc file

Customizing the Hierarchical Netlister (HNL)

How to Define Power Node and Ground Node?

You can define `powerNets` and `groundNets` in the `.simrc` file. For example, if you enter the following lines in your `.simrc` file

```
powerNets  = '("VCC!")
groundNets = '("GND!" "gnd!" )
```

The auCdl netlist will show the following line:

```
*.GLOBAL VCC!:P GND!:G gnd!:G
```

You can use the `auCdlSkipMEGA` flag for conditional printing of the `*.MEGA` statement in the auCdl netlist.This flag can be placed in the `.simrc` file, which is read by the netlister.

The `auCdlSkipMEGA` flag is used as follows:

```
auCdlSkipMEGA = ’nil
```

This is the default value. This enables printing of the statement in the netlist.

```
auCdlSkipMEGA = ’t
```

When set, the `*.MEGA` statement is not printed in the auCdl netlist.

How to Evaluate Expressions during auCdl Netlisting?

You might want to evaluate design variables that have been copied to the cellview using ADE and whose values are needed during verification. The Analog Expression Language mode using which auCdl evaluates expressions is determined by the setting of the SKILL environmental flag `auCdlSetTopLevelEvalMode`. Its valid values are `'t` and `nil`. The default value is `nil` and it causes auCdl to evaluate expressions by using inheritance operators. You can change the mode to full evaluation by setting the value of this flag to `'t`.

How to Preserve Devices in a Netlist?

The `si.env` file defines the following variables that determine if resistors, capacitors, diodes, or all devices must be preserved in the netlist.

```
preserveRES       preserveCAP

preserveDIO       preserveALL
```

How to Print CDL Commands?

The following variables let you print the associated CDL commands.

```
checkRESVAL       checkDIOAREA
checkCAPVAL       displayPININFO
checkDIOPERI      shortRES
checkRESSIZE      resistorModel
checkCAPAREA
```

How to Remove Devices from a Netlist?

During hierarchical netlisting, you can short the terminals of a device and replace that device with a surviving net. For example, you can short terminals of parasitic devices to remove them from the netlist.

The terminals of a device can be short using any of the following methods:

- Setting the lxRemoveDevice string property at the instance level.
- Using the `hnlUserMultiTermShortCVList` SKILL variable.
- Using the `hnlUserShortCVList` SKILL variable.

Removing Devices with Two Terminals

Removing Devices with Multiple Terminals

How to Set Default View List, Stop List, Netlist Type, and Comments?

You can use the following variables to define the standard view list, stop list, and netlist type and specify the value of the print comments flag.

Variable	Description
`cdlSimViewList`	A list of views. The default is `'("auCdl" "schematic")`.
`cdlSimStopList`	A list of views. The default is `'("auCdl")`.

`cdlNetlistType`	Netlist type hierarchical (`'hnl`) or flat (`'fnl`). The default is `'hnl`.
`cdlPrintComments`	Print comments? Yes (`'t`) or no (`'nil`). The default is `'nil`.

The following variables are used for instance-based switch list configuration and also can be set:

`simInstViewListTable`

`simInstStopListTable`

How are Global Power and Ground Signals Defined?

You can now use the CDL Out form to declare global power signal and global ground signals by following the steps given below:

1. In the CIW, choose *File – Export – CDL*.
2. In the fields, *Global Power Signals* field and Global *Ground Signals*, enter signal names respectively.

The values that you enter using the form will be added to `*.GLOBAL` and `*.PIN` statement.

`:G` and `:P` will be appended to the signal names based on the nets presence in the variables `simPowerNets` and `simGroundNets` in `.simrc` file.

How Are NLP Expressions Used during auCdl Netlisting?

Netlisting Properties (NLP) expressions provide support for user defined properties in auCDL netlisting. You can use different NLP expressions depending on your requirements. Details about each NLP expression is described below:

- NLP expression beginning with "`[+`" is equivalent to `pPar` in AEL expression. For example, if property "`myprop`" has value "`[+subProp]`", it will appear in auCDL netlist as `myProp = subProp`. The netlister prints the value of `subProp` for the `SUBCKT` on which it is defined.
- NLP expression beginning with "`[@`" is equivalent to `atPar` in AEL expression. For example, if property "`myprop`" has value "`[@subProp]`", it will appear in auCDL netlist as `myProp = subProp`.

- NLP expression beginning with “`[~`” is equivalent to `iPar` in AEL expression. For example, if property “`myprop`” has value “`[~subProp:new value %: not found]`” and `subProp` has a value of 10 for the instance being netlisted, it will be printed in the netlist as `myProp = new value 10`. However, if `subProp` is not defined at instance level, it will be printed in the netlist as `myProp = not found`.

How are Global Pins Mapped during auCdl Netlisting?

In the DFII environment, global signals in a netlist end with a ! character. If you do not want global signals to end with `!`, you can specify this by using either one of the following methods:

- Click *File – Export – CDL* to open the CDL Out form and select the *Map Pin Names from <> to []* option button.
- In the `.simrc` file, set the SKILL environmental variable pinMap to `'t`. This is a boolean variable and can have the value `'t` or `'nil`. This variable when set to `'t` uses the following rules to map net names:

```
"+" -> nil
"(" -> nil
")" -> nil
"," -> nil
"/" -> nil
"." -> nil
$" -> nil
"[" -> nil
"]" -> nil
"<" -> "["
">" -> "]"
"!" -> nil
```

Default direction of global power and ground pins is `INPUT`.

The SKILL environmental variable `hnlMapNetInName` can be used similarly. For example:

```
pinMap = 't
```

is equivalent to:

```
hnlMapNetInName = list('("+" nil) '("(" nil) '(")" nil) '("," nil) '("/" nil) '("."
nil) '("$" nil) '("[" nil) '("]" nil) '("<" "[") '(">" "]") '("!" nil) )
```

How to Rename Cell Names during auCdl Netlisting?

You can define the auCdlModuleNameMapFunc SKILL variable in the .simrc file to rename cell names in the auCdl netlist.

For example, to add a prefix AAA_ to the cell names in the auCdl netlist, add the following entries in the .simrc file:

```
auCdlModuleNameMapFunc = ’myPoCellNameMap
procedure(myPoCellNameMap( cvID )
poCellNameMap( cvID~>libName cvID~>cellName cvID~>viewName)
         )
procedure(poCellNameMap(lib cell view)
   prog((mapname)
   sprintf(mapname "AAA_%s"  cell)
   return(mapname)
       )
```

⚠ Setting this variable does not detect the cell name collision and the netlister uses the same name if two cells have the same name (even if they are in different `subckt`).

How to Rename Pcell Subcircuits in a auCdl Netlist?

You can define the `nlSetPcellName` SKILL procedure in the `.simrc` file to customize renaming of Pcell subcircuits in the auCdl netlist.

For more information about the `nlSetPcellName` procedure, see Virtuoso Analog Design Environment SKILL Language Reference.

How to Customize Bulk Node Search during auCdl Netlisting?

Bulk node connection or the substrate connection on a device is specified by using the CDF property of the form (`progn bn`). If the bulk node property is found on the instance terminal, then the net name connected to the instance terminal, which has the same name as the value of bulk node property on the instance, is printed as the bulk node connection in the netlist.

When the bulk node property is found on both instance terminal and cellview terminal, then by default, preference is given to the instance terminal. However, in this case, user can also specify how the bulk node property should be selected. For example, user can give preference to cellview terminal over instance terminal for bulk node connection by adding the following in the `.simrc` file:

```
auCdl.bulkNodeLookUp = '("cvTerm")
```

⚠ `cvTerm` can be used only when `bulknode` is found on both instance terminals and cellview terminals.