Virtuoso Power Manager Guide

Product Version ICADVM20.1
October 2020

Printed in the United States of America.

Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

Contents

Preface

The guide describes how to use Virtuoso® Power Manager, which lets you specify the power intent for designs. It provides enhanced support for capturing, automatically extracting, exporting, and verifying the power intent of IP designs. You can run a set of predefined checks to identify typical design problems, which can be related to voltage domains, power domains and connectivity, or biasing issues. It enables you to define the power intent for both analog and digital sub-blocks.

This preface contains the following topics:

Scope

The functionality described in this guide can be used only in ICADVM20.1 advanced nodes and advanced methodologies releases.

Licensing Requirements

You need the following licenses to use Power Manager.

Base Product Name	License Feature Name	License Type
Virtuoso Schematic Editor XL	`Virtuoso_Schematic_Editor_XL` (95115)	UHD
Virtuoso Power Manager	`VIRTUOSO_POWER_MANAGER` (95127)	J

Note: The VIRTUOSO_POWER_MANAGER license is used until the last Power Manager schematic window along with the Power Manager Setup form is closed.

Related Documentation

What's New and KPNS

- *Virtuoso Power Manager What's New*
- *Virtuoso Power Manager Known Problems and Solutions*

Installation, Environment, and Infrastructure

- *Cadence Installation Guide*
- *Virtuoso Software Licensing and Configuration User Guide*
- *Virtuoso Design Environment User Guide*
- *Cadence Application Infrastructure User Guide*
- *Virtuoso Design Environment SKILL Reference*
- *Virtuoso Schematic Editor SKILL Reference*
- *IEEE 1801 User Guide*
- *Conformal Low Power User Guide*
- *Low-Power Simulation Guide (IEEE 1801)*

Technology Information

- *Virtuoso Technology Data User Guide*
- *Virtuoso Technology Data ASCII Files Reference*
- *Virtuoso Technology Data SKILL Reference*

Virtuoso Tools

- *Virtuoso Schematic Editor User Guide*
- *Virtuoso ADE Assembler User Guide*
- *Virtuoso ADE Explorer User Guide*
- *Virtuoso UltraSim Simulator User Guide*

- *Virtuoso AMS Designer Simulator User Guide*
- *Spectre Circuit Simulator and Accelerated Parallel Simulator RF Analysis in ADE Explorer User Guide*
- *Spectre Circuit Simulator and Accelerated Parallel Simulator User Guide*
- *Spectre Circuit Simulator Reference*

Additional Learning Resources

Video Library

The Video Library on the Cadence Online Support website provides a comprehensive list of videos on various Cadence products.

To view a list of videos related to a specific product, you can use the *Filter Results* feature available in the pane on the left. For example, click the *Virtuoso Layout Suite* product link to view a list of videos available for the product.

You can also save your product preferences in the Product Selection form, which opens when you click the *Edit* icon located next to *My Products*.

Virtuoso Videos Book

You can access certain videos directly from Cadence Help. To learn more about this feature and to access the list of available videos, see Virtuoso Videos.

Rapid Adoption Kits

Cadence provides a number of Rapid Adoption Kits that demonstrate how to use Virtuoso applications in your design flows. These kits contain design databases and instructions on how to run the design flow.

In addition, Cadence offers the following training courses on Virtuoso Schematic Editor:

- Virtuoso Schematic Editor
- Virtuoso Analog Design Environment
- Using Virtuoso Constraints Effectively

- Virtuoso Spectre Circuit Simulator
- Spectre Simulations Using Virtuoso ADE
- Virtuoso Electrically-Aware Design with Layout Dependent Effects

To explore the full range of training courses provided by Cadence in your region, visit Cadence Training or write to `training_enroll@cadence.com`.

Note: The links in this section open in a separate web browser window when clicked in Cadence Help.

Help and Support Facilities

Virtuoso offers several built-in features to let you access help and support directly from the software.

- The Virtuoso *Help* menu provides consistent help system access across Virtuoso tools and applications. The standard Virtuoso *Help* menu lets you access the most useful help and support resources from the Cadence support and corporate websites directly from the CIW or any Virtuoso application.
- The Virtuoso Welcome Page is a self-help launch pad offering access to a host of useful knowledge resources, including quick links to content available within the Virtuoso installation as well as to other popular online content.

  The Welcome Page is displayed by default when you open Cadence Help in standalone mode from a Virtuoso installation. You can also access it at any time by selecting *Help – Virtuoso Documentation Library* from any application window, or by clicking the *Home* button on the Cadence Help toolbar (provided you have not set a custom home page).

For more information, see Getting Help in *Virtuoso Design Environment User Guide.*

Customer Support

For assistance with Cadence products:

- Contact Cadence Customer Support

  Cadence is committed to keeping your design teams productive by providing answers to technical questions and to any queries about the latest software updates and training needs. For more information, visit https://www.cadence.com/support.

- Log on to Cadence Online Support

Customers with a maintenance contract with Cadence can obtain the latest information about various tools at https://support.cadence.com.

Feedback about Documentation

You can contact Cadence Customer Support to open a service request if you:

- Find erroneous information in a product manual
- Cannot find in a product manual the information you are looking for
- Face an issue while accessing documentation by using Cadence Help

You can also submit feedback by using the following methods:

- In the Cadence Help window, click the *Feedback* button and follow instructions.
- On the Cadence Online Support Product Manuals page, select the required product and submit your feedback by using the *Provide Feedback* box.

Understanding Cadence SKILL

Cadence SKILL is a high-level, interactive programming language based on the popular artificial intelligence language, LISP. It lets you customize and extend your design environment. Using SKILL you can validate the steps of your algorithm incrementally before incorporating them into a larger program.

For more information about the SKILL language, see Getting Started in the *SKILL Language User Guide*.

Using SKILL Code Examples

The SKILL APIs in this user manual are explained with illustrative code examples.

You can copy these examples from the manual and paste them directly into the Command Interpreter Window (CIW) or use the code in non-graphical SKILL mode.

Sample SKILL Code

The following code sample shows the syntax of a SKILL API that accepts three arguments.

The first argument `t_sessionName` is a required argument, where `t` signifies the data type of the argument. The second and third arguments `?optionName t_optionName` and `?historyName t_historyName` are optional keyword arguments (identified by a question mark), which are specified in name-value pairs and can be placed in any order during the function call.

The return value is the value that the SKILL API returns after evaluating the expression. In this case, it is a list of status values, `l_statusValues`.

Accessing API Help

Quick reference information for SKILL APIs is available from the CIW and the SKILL API Finder. To access the reference information for a particular SKILL API, do one of the following:

- Type `help <function_name>` in the CIW.
- Type `startFinder ( [ ?funcName t_functionName ] )` in the CIW.

- Start the SKILL API Finder from the CIW by choosing *Tools – Finder* or type `cdsFinder` on the UNIX command line.

  In the *Search in* field of the displayed Cadence SKILL API Finder window, type the SKILL API name for which you want to display the help information and click *Go*.

  The matches for the searched SKILL API appear in the *Results* area.

- To view the complete documentation of the searched SKILL API, select the API name in the *Results* area and click the *More Info* button. The complete documentation of the selected SKILL API appears in a new Cadence Help window.

Typographic and Syntax Conventions

The following typographic and syntax conventions are used in this manual.

text	Indicates names of manuals, menu commands, buttons, and fields.
`text`	Indicates text that you must type as presented. Typically used to denote command, function, routine, or argument names that must be typed literally.
`z_argument`	Indicates text that you must replace with an appropriate argument value. The prefix (in this example, *`z_`*) indicates the data type the argument can accept and must not be typed.
|	Separates a choice of options.
`{ }`	Encloses a list of choices, separated by vertical bars, from which you **must** choose one.
`[ ]`	Encloses an optional argument or a list of choices separated by vertical bars, from which you **may** choose one.
`[ ?argName` *`t_arg`* `]`	Denotes a *key argument*. The question mark and argument name must be typed as they appear in the syntax and must be followed by the required value for that argument.
`...`	Indicates that you can repeat the previous argument. Used with brackets to indicate that you can specify zero or more arguments. Used without brackets to indicate that you must specify at least one argument.
`,...`	Indicates that multiple arguments must be separated by commas.
`=>`	Indicates the values returned by a Cadence® SKILL® language function.
`/`	Separates the values that can be returned by a Cadence SKILL language function.

If a command-line or SKILL expression is too long to fit within the paragraph margins of this document, the remainder of the expression is moved to the next line and indented. In code excerpts, a backslash (\ ) indicates that the current line continues on to the next line.

Identifiers Used to Denote Data Types

Data type identifiers are used to indicate the type of value required by an API argument. These data types are denoted by a single letter that is prefixed to the argument label and is separated from the argument by an underscore; for example, *t* is the data type in *t_viewName*. Data types and underscores are used only as identifiers; they must not be typed when specifying the argument in a function.

Prefix	Internal Name	Data Type
a	array	array
A	amsobject	AMS object
b	ddUserType	DDPI object
B	ddCatUserType	DDPI category object
C	opfcontext	OPF context
d	dbobject	Cadence database object (CDBA)
e	envobj	environment
f	flonum	floating-point number
F	opffile	OPF file
g	general	any data type
G	gdmSpecIlUserType	generic design management (GDM) spec object
h	hdbobject	hierarchical database configuration object
I	dbgenobject	CDB generator object
K	mapiobject	MAPI object
l	list	linked list
L	tc	Technology file time stamp
m	nmpIlUserType	nmpIl user type
M	cdsEvalObject	cdsEvalObject
n	number	integer or floating-point number
o	userType	user-defined type (other)
p	port	I/O port
q	gdmspecListIlUserType	gdm spec list

Prefix	Internal Name	Data Type
r	defstruct	defstruct
R	rodObj	relative object design (ROD) object
s	symbol	symbol
S	stringSymbol	symbol or character string
t	string	character string (text)
T	txobject	transient object
u	function	function object, either the name of a function (symbol) or a lambda function body (list)
U	funobj	function object
v	hdbpath	hdbpath
w	wtype	window type
sw	swtype	subtype session window
dw	dwtype	subtype dockable window
x	integer	integer number
y	binary	binary function
&	pointer	pointer type

For more information, see *Cadence SKILL Language User Guide*.

1

Getting Started with Virtuoso Power Manager

Virtuoso Power Manager provides an interface to specify, import, and export low power intent for designs. It provides the capability to perform static low power verification on designs by using Conformal Low Power (CLP) integration with Virtuoso and supports analog, digital, and mixed-signal implementations.

Power Manager, by using the import flow, can annotate and stitch the top-down power connectivity for a hierarchical design by using the connectivity model for inherited connections. Using the export flow, you can extract the power intent of a fully connected design schematic. You can also generate a Liberty-based macro cell for custom blocks using Power Manager for full-chip power intent verification.

When using special power-saving techniques for efficient power consumption, you often implement stringent power schemes throughout the design that add to the overall complexity while verifying the design. In addition, there are design scenarios that should be verified for the correct functioning of a circuit, for example, the correct biasing of MOS devices. To reduce the complexity, you should be able to perform a few basic-to advanced-level circuit design checks on schematic designs. The In-Design Checks feature in Power Manager provides such static checks that do not require circuit simulation.

The chapter includes the following sections:

Introduction to Power Manager

An important consideration in IC design is to conserve power. Designers use special power saving techniques to achieve this objective. Some examples of these techniques include:

- Supplying different voltages for different groups of cells, therefore, creating multiple power domains.
- Using low power special cells to switch between the power-supply levels or to shut off groups of cells during specific periods of circuit activity, in turn, creating different power modes.

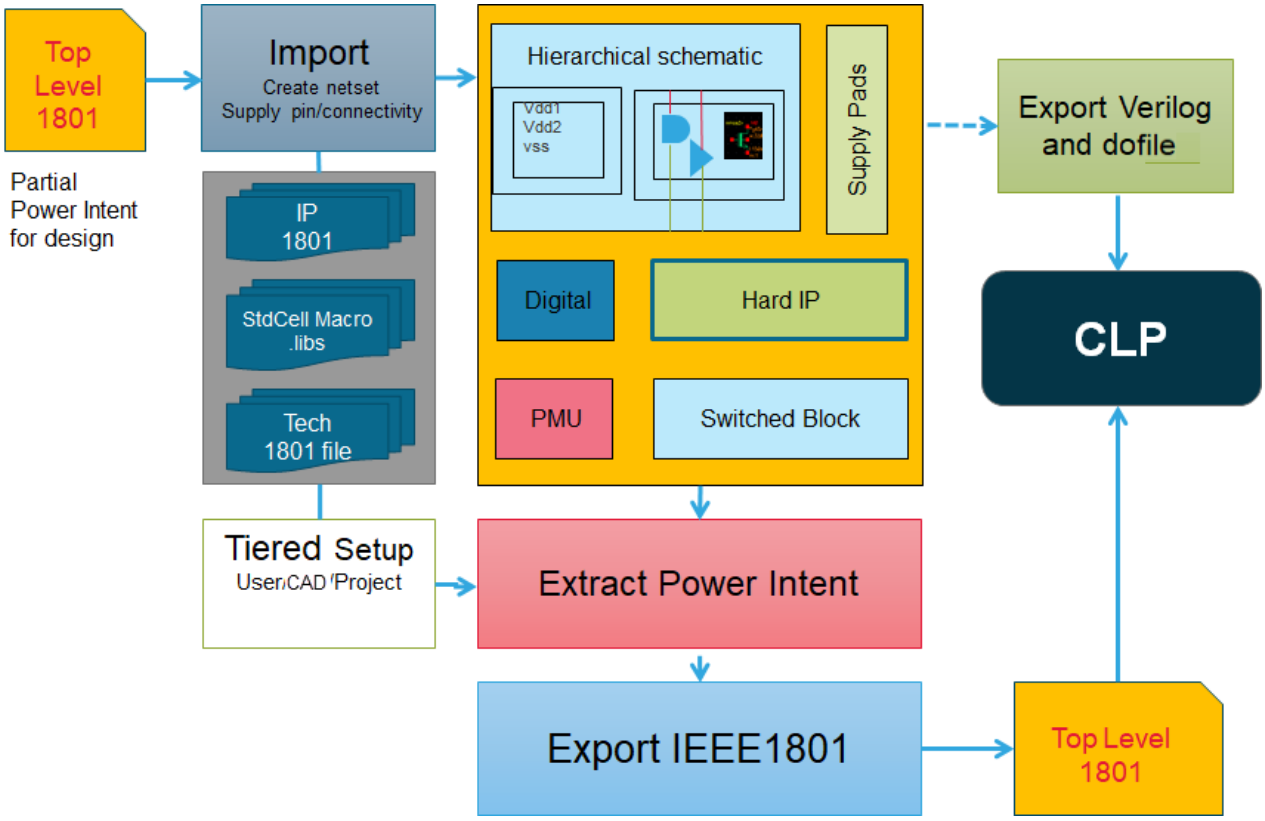
The definitions of the power domains, power modes, and low power special cells specify the power intent for a design. When you create a design that has some low power design techniques applied, you should be able to:

- Package and export the power intent along with the design by extracting the same from a hierarchical schematic.
- Verify the power intent of the design along with the functionality before it is further instantiated in other designs.
- Import an existing power intent and annotate it on the hierarchical schematic in terms of a completely resolved top-down power connectivity.

Power Manager addresses the need for the creation and verification of power-aware designs, including validating power intent changes during IP authoring. It provides the capability to annotate the design's power intent-related issues on the schematic. Therefore, it helps in the timely correction of power-related violations in the design.

Use Power Manager to specify, import, extract, and export the low power intent for designs. You can capture the power intent for the design (including its submodules) and map the power

domains of an IP block with the domains of the designs. This helps in integrating an IP in the design.



Power intent definition at the design level is needed to connect IP blocks, which could be empty, partially complete, or non-structural, to top-level power ground (PG) pins. This definition, along with incremental change in power intent resulting from design edits, can be exported.

Low power verification is required to ensure structural and electrical accuracy for mixed-signal designs. Native low power checks within the Virtuoso Power Manager environment help in detecting low power violations during the IP authoring stage. The early feedback saves iterations later in the cycle. In-design checks work at the device level and enable the structural verification of custom AMS blocks. These are static circuit checks, which can be performed to fix design issues before proceeding to the dynamic verification (simulation) stage. The checks are performed after voltage propagation is done on the topology of a circuit. The voltage propagation does not require a transient simulation; therefore, a static check is faster than a dynamic check.

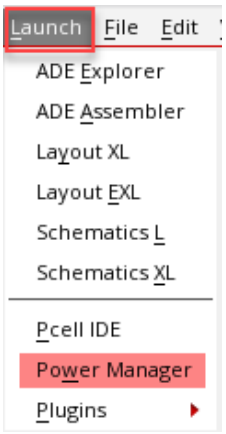
In-design checks are targeted to provide hints to designers about various low power and structural discrepancies in the design and help in improving the efficiency of the design development cycle by reporting design issues that can be fixed while the schematic design

is still under development.

Verification can happen early in the design cycle and natively within the Virtuoso environment without invoking other tools; therefore, no data sharing or data translation occurs between tools. All formats are supported for checks, such as 1801, Liberty, or dotlib. Inherited and global power supply connections are also supported. In-Design Checks are applicable to discrete devices. The checks also work on hierarchical and read-only designs.

Launching Power Manager

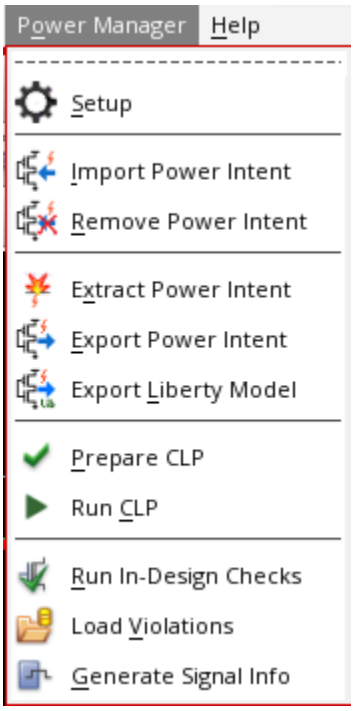
To start Virtuoso Power Manager, click *Launch – Power Manager*.



The Power Manager workspace window is opened.

Power Manager GUI

In the Power Manager workspace, the following menu items are available in the *Power Manager* menu.



Alongside, these options are also available in the Power Manager toolbar in the workspace. The toolbar lets you access the various commands and related forms supported by Power Manager. It is dockable and can be placed at the desired location within the window.

Refer to the following table for more information about the options on the toolbar.

Icon	Command	Use to...	Form
	Setup	Specify the setup information for running In-Design checks, importing, exporting, extracting, or removing power intent.	Specifying the Setup
	Import Power Intent	Import the power intent for the given cellview.	Importing the Power Intent of a Design

Icon	Command	Use to...	Form
	Remove Power Intent	Remove all the power intent information from the current cellview.	Removing the Imported Power Intent
	Extract Power Intent	Extract the power intent of the given cellview.	Extracting the Power Intent from a Design
	Export Power Intent	Export the power intent of the current cellview by using a `.upf` file.	Exporting 1801 Power Model
	Export Liberty Model	Export the Liberty model of the current cellview by using a `.lib` file.	Exporting Liberty Power Model
	Prepare CLP	Generate the input files at a user-specified location and therefore, provide an opportunity to view, inspect, and edit the files.	Preparing and Running CLP
	Run CLP	Start CLP and use the files created during CLP preparation, for power verification	Preparing and Running CLP
	Run In-Design Checks	Perform static checks on the given design.	Running Checks
	Load Violations	Load the file containing violations after the design check.	Loading the Violations Database
	Generate Signal Info	Generate an output file that contains the net voltages for supply and signal nets, which are computed based on supply net and technology information provided in the Power Manager setup.	Generating Signal Information

2

Virtuoso Power Manager Flows

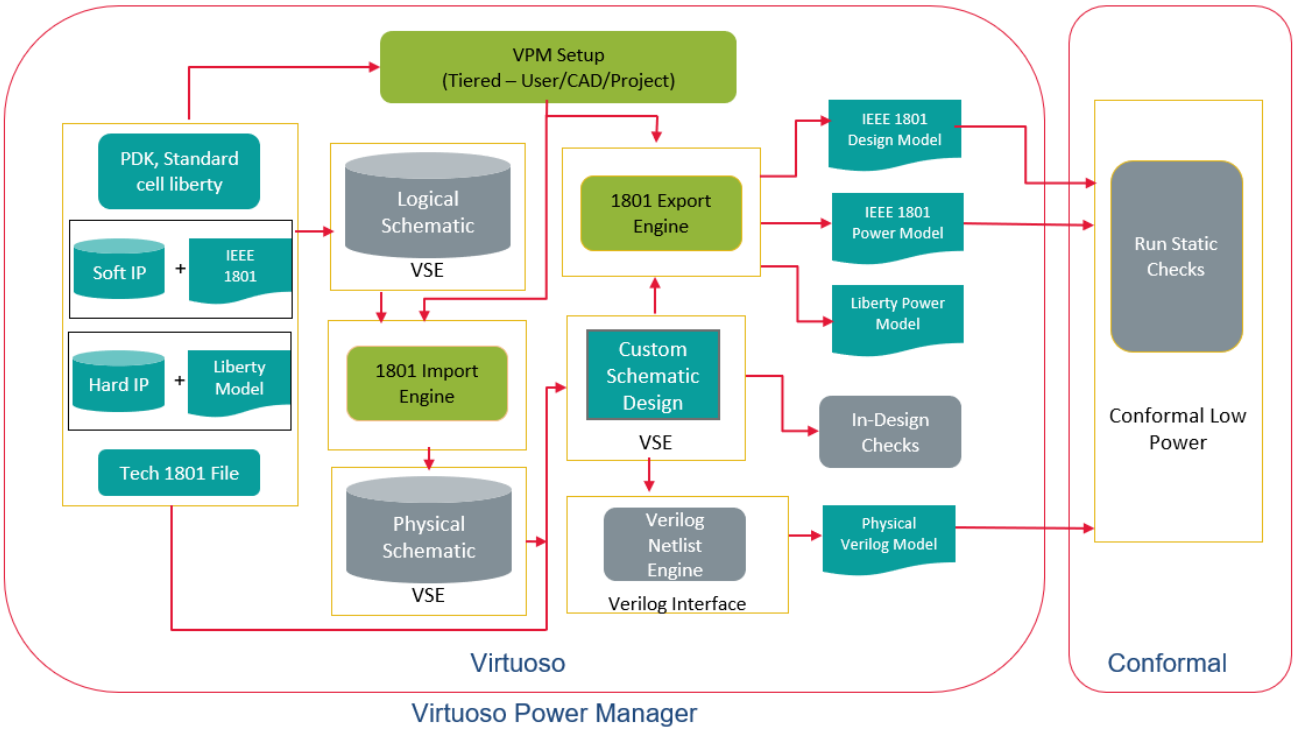
The flows in Virtuoso Power Manager offer an automated solution for capturing the power intent of a design. Usually for small designs, it is convenient to write the power intent manually. As the design complexity increases, it becomes mandatory to have an automated solution. The complexity in designs can be about multiple power domains, different voltage levels, switchable power domains, or the functionality implemented using low power design techniques by the use of low power special cells, such as level shifters, isolation cells, and power switches,

You can seamlessly capture the power intent of digital, analog, and mixed-signal designs and verify that the power intent has been implemented correctly. In addition, you can also track power intent changes during IP authoring. You can export the power intent of the IP design in the standard IEEE 1801 format, which can be further utilized by the SOC verification team for full-chip low power verification.

The chapter includes the following sections:

Basic Flow in Power Manager

The basic Power Manager flow is illustrated below.



The Power Manager flow includes the following main tasks:

- Prepare setup for capturing power intent.
- Import existing 1801 power intent on a hierarchical schematic.
- Update design.
- Run In-Design Checks.
- Extract relevant power intent information from design.
- Export the extracted design information in the IEEE 1801format.
- Verify the exported power intent of the design.

Recommended Use Model for Power Intent Creation and Verification

Follow the use model described by the steps given below to get an accurate power intent for your design:

1. Prepare a setup file for importing power intent on a design, running In-Design Checks, or extracting power intent from a design.

   This requires registration of the components that describe power intent. For example, you can register power nets, ground nets, and special low power cells.

   For more details, refer to Setup for Automatic Extraction of Power Intent.

2. Register special low power cells or custom standard cells by using the Liberty definitions or a 1801 special cell definition file.

   For more details about how to register low power special cells, refer to Registration of Low Power Special Cells and Standard Cells.

3. If there is an incomplete hierarchical design that already has the power intent available, for example, a digital block, use the import 1801 flow to complete the power connectivity of that hierarchical design.

   Perform the schematic hierarchy check while importing an existing 1801 power specification on a hierarchical schematic.

   For details, refer to Importing the Power Intent of a Design.

4. Open the design in Power Manager and check for an existence of primitive cell instances or single supply cells, such as transistors or resistors at the top level of designs. If these cells exist at the top level, create one or more new blocks and move these cells to the new blocks.

   This is required because the primitive instances do not have a power pin or ground pin and therefore, are not assigned to any power domain or domain mapping. In such a case, the design netlist generated for the verification includes these primitive cell instances, but the exported 1801 file does not include these instances. As a result, during power intent verification, when CLP does not find the power intent for these instances in the1801 file, it reports errors or warnings. These can be avoided during netlisting by using the appropriate flags to customize the netlist so that it does not extract primitives.

   For details, refer to Verifying Power Intent of a Design.

5. Load the setup file, including the customizations required for running the In-Design checks. Check the analog or mixed signal custom blocks using static, pre-configured In-Design Checks, and verify the low power design implementation.

For more details, refer to Running In-Design Checks.

Once the results obtained are satisfactory, extract the power intent details of the top design. Power Manager extracts the power intent of hierarchical design, which can be exported to a 1801 file. A separate OA view called the power view is also generated at the time of extraction that captures all the setup information, internal data structure, and the power intent details. This view can be reused in the same Virtuoso session.

For more details, see Exporting Power Intent of a Design.

6. The standard and special cells in the design should have their associated schematic or Verilog Symbol views with the PG pin information. This is a requirement for the tool to generate a Verilog netlist with the PG information, before verifying the power intent.

Tip

CLP is a digital verification tool. It supports the inherited connections and global nets with explicit p/g connections to the standard cells because it needs maximum low power check coverage. CLP performs physical low power verification by tracing the explicit p/g nets and deriving power domain information accordingly. This is the most comprehensive design structural check.

7. Verify the power intent of your design.

   For more details, refer to Verifying Power Intent of a Design.

Caution

If the top design is large, the power intent extraction step might lead to memory-related issues. In such cases, follow the bottom-up approach, where you first extract the power intent of sub-blocks separately in 1801 files and then, associate the 1801 files with their respective instances in the top design. Consequently, when you extract power intent for the top design, the tool consumes less memory. Also, you can apply the bottom-up approach when extraction options cannot be shared between the top design and the sub-blocks. For example, when the power / ground nets definitions, power / ground net voltage definitions, or power net–ground net voltage pair definitions in the top design are different from the sub-block definitions, use the bottom-up approach.

3

Setup for Automatic Extraction of Power Intent

To run In-Design Checks or to extract the power intent from a design, you need to provide inputs that are required by the tool for correct identification of design topology and define key set of rules. For example, you can define a setup that identifies all the power nets in your design, creates power domains, creates supply states, or redefines the severity of certain checks.

The chapter includes the following sections:

Introduction

The Power Manager setup can be classified as:

- Common setup: This type of setup information is common to a set of designs. It includes technology, environment, data integrity checks, and so on. This setup information is stored in a CSF-searchable location and is applied when no design-specific information is found.
- Project-specific setup: This information consists of the project-specific setup and design-specific details. It includes all types of settings and it is automatically applied if available for a design. You can specify the settings in the Power Manager Setup form and save them in the setup file for a project to ensure that each command is implemented according to the project setup.

A setup file can be placed at one of the following three locations:

- Device-level information inside PDK Library in `lpLibSetup.il`
- Project-level common settings at the CAD level inside `.cadence/dfII/vpm` or some other CSF-compatible path in `lpSetup.il`
- User-defined settings that are defined in Power Manger Setup form and saved in a setup template file

There are two options in the setup template to control the enabling of the project-level (CAD) settings in the final merged setup loaded for Power Manager.

1. `loadEnvironmentOptions`:

   Controls whether the `.cadence/dfII/vpm/lpSetup.il` file is loaded automatically as a part of the final merged setup.

   - If set to `true`, the project-level settings are considered as a part of the final setup.
   - If set to `nil`, the project level settings are ignored for the final setup.
   - If this option is not specified in the Power Manager setup template, it is assumed to be true and is considered for the final setup.

2. `pushUnmodifiedEnvOptions`:

   Controls the reading of the project-level (CAD) settings, at a later stage. For example, if there is any update in the project-level settings and the changes need to be applied in the final merged setup for the Power Manager.

   - If set to `true`, any changes done in the project-level settings are read and updated.
   - If set to `nil`, any changes done in the project level settings are ignored in the final setup.

You can define these options in the setup if you want to read the project-level setup initially or always want to ensure that the user-defined setup is synchronized with the project-level (CAD) setup.

The final setup is the result of merging all the three setups, which can be verified by exporting in the file:

```
vpmExportPowerIntentSetup("random_computation_flat" "random_computation_top"
"schematic" "lpSetup_export.il" ?filter "allSetupOptions" ?overwriteExisting t)
```

Currently, if there are common settings in the setup file, for example, supply nets are specified in CAD and user-defined setup, the user-defined setup has a higher precedence.

Specifying the Setup

This section describes the various input attributes that are required in the setup file template, considering all the flows supported by Power Manager. You can register different attributes that are required for different flows in the Power Manager Setup form. This setup can be saved eventually in a setup file template.

The Power Manager Setup form gives you the flexibility to dynamically switch and update more than one design for power information. However, it is recommended to update one design at a time, close the form, and reopen it for updating the power intent information for a different design. This ensures that the setup information is not inadvertently lost when switching from the setup for one design to the other.

The section includes the following sub-sections:

- Supply Nets Registration
- Library Registration
- Device and Cell Registration
- In-Design Checks
- Supply Set and Power Domain Registration
- Miscellaneous Settings

Supply Nets Registration

The supply nets registration information across the design hierarchy can be registered in the following ways by using the Power Manager setup form:

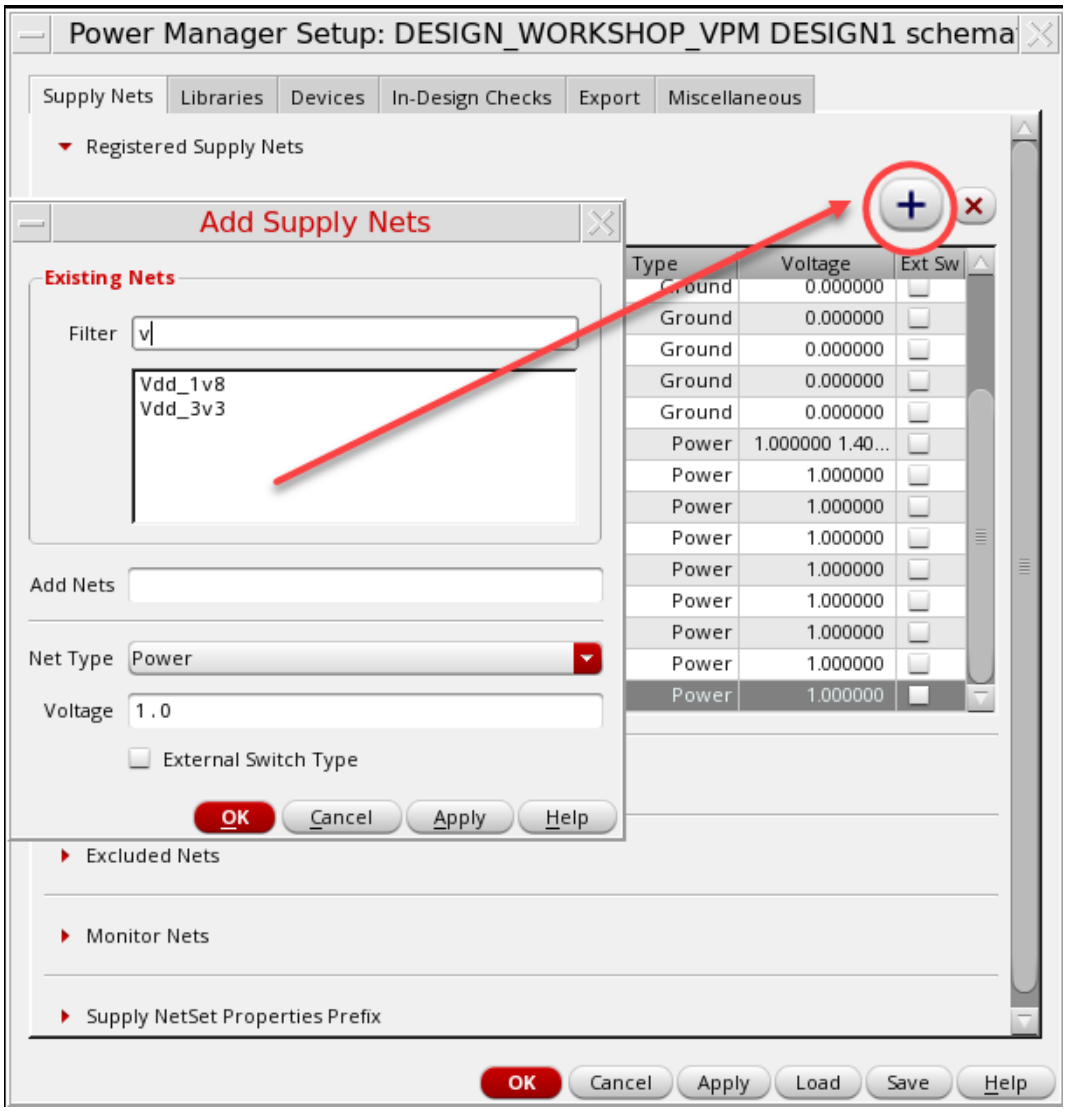
- Name-Based Supply Nets Registration
- Regular Expressions-Based Supply Nets Registration
- Supply NetSet Properties Prefix Registration
- Power and Ground Nets Exclusion from Name-Based Registration

Name-Based Supply Nets Registration

You can set a rule to register certain names as names of power nets or ground nets. If a net in your design has one of the registered names, the net is identified accordingly. It is recommended to follow a similar naming convention for supply net naming across the design hierarchy. Here are the steps to register the supply nets.

1. Double-click the fields in the *Registered Supply Nets* table to edit. To add the supply nets, click  . It opens the Add Supply Nets form.
2. Specify the values in the form for Supply Nets Registration.

3. Click  to remove added supply nets. Press `Shift + Ctrl` to select nets and register them as power or ground. Simultaneous registration of existing and user-defined nets as power/ground is possible.



Important

The existing nets shown in the Add Supply Nets form are the nets corresponding to the top-level ports of the design in consideration.

You need to specify the voltage values of the net for the top-level supply nets. You also need to register internal nets, such as low drop out (LDO) nets or output of a voltage divider, and

provide the voltage values for such internal nets. For a hierarchical design, the voltage values specified from the top-level nets are propagated to the levels below in the following ways:

- If the cells used in the design do not have their associated Liberty file, the voltage propagation for them happens hierarchically as per the top-level voltage values.
- If the cells used in the design have their associated Liberty/1801 cell bindings, the voltage map (from Liberty) or power states (from 1801) are expected to match the voltage levels specified for top level supplies.

You can also define multiple voltages for a top-level net. If you define multiple voltages for both power nets and ground nets, Power Manager considers all possible supply states based on all the supply net voltages provided in the setup. In-Design Checks consider all possible supply states based on all the supply net voltages provided in the setup information. The supply states are created depending upon the unique voltage values for a power net–ground net pair.

Consider a voltage specification with supplies `VDD`, `VSS`, and `VDD1V2` where `VDD` is 0.8V and 1.0V, `VSS` = 0.0V and 0.4V, and `VDD1V2`=1.2v.

```
netVoltages (("VDD" (0.8 1.0))
("VDD1V2" (1.2))
("VSS" (0.0 0.4)))
```

The following power modes are created:

Supplies	**VDD**	**VDD1V2**	**VSS**
State	OFF	OFF	OFF
	ON(0.8V)	ON(1.2V)	ON(0V)
	ON(0.8V)	ON(1.2V)	ON(0.4V)
	ON(1.0V)	ON(1.2V)	ON(0V)
	ON(1.0V)	ON(1.2V)	ON(0.4V)

Note: You must provide the supply net voltage values for creating the power states based on the switching behavior of the corresponding supply nets.

Note: Specifying net voltages in the setup is not mandatory if the import flow is used.

Regular Expressions-Based Supply Nets Registration

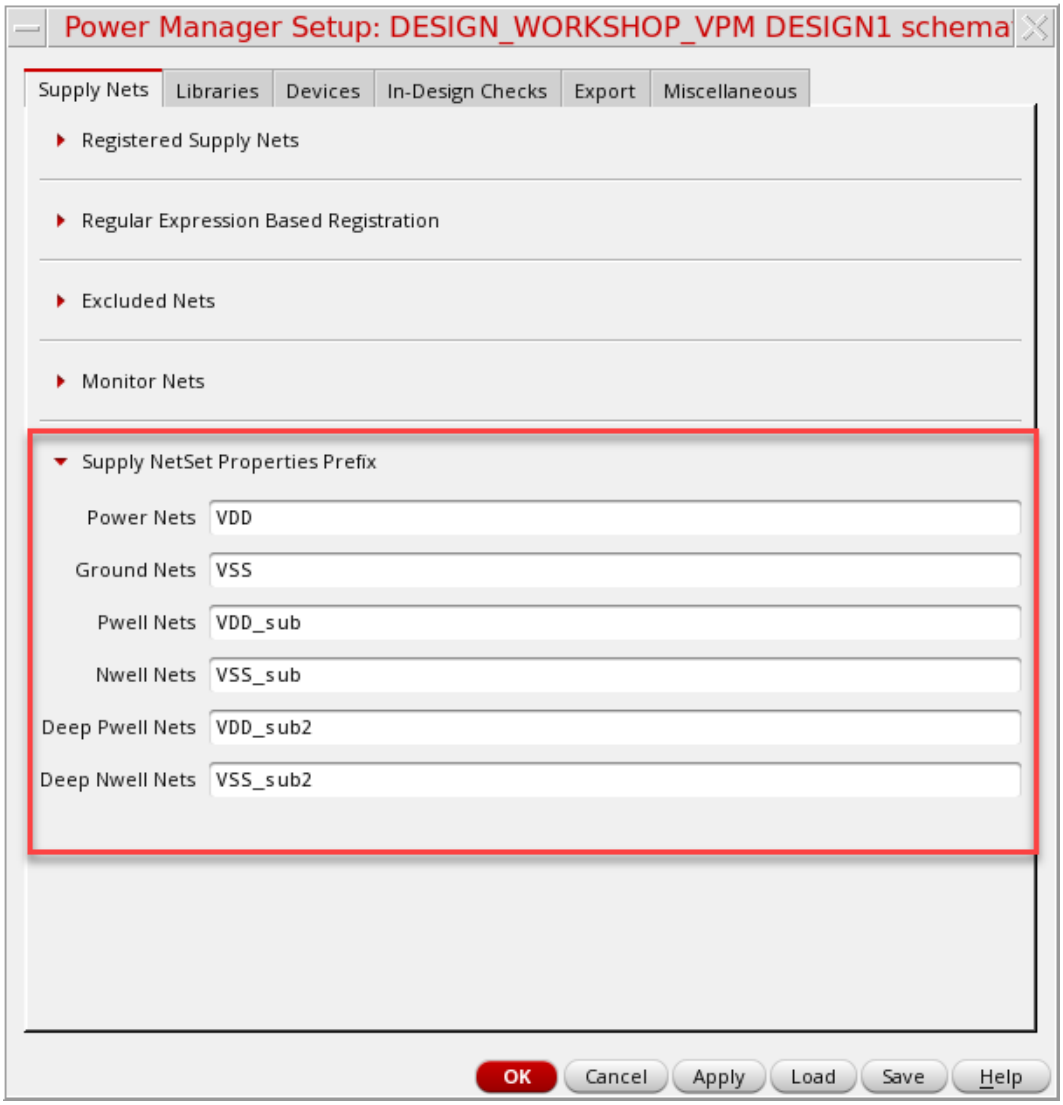
If you do not know the names of the nets that exist down the hierarchy, there are some naming conventions followed. You can specify regular expressions for nets in the Power Net Regular Expressions or Ground Net Regular Expressions fields. For example, if you add `vdd` as a regular expression, such as `iovdd`, `avddd1`, or any net name that contains this is considered as power net. Similarly, if `GND` is specified as a regular expression, names such as `GNDA`, `GNDD`, and so on, are considered as ground nets. Also, if `[v][V]`, `[d][D]`, and `[d][D]` are defined as regular expressions, all combinations are considered, such as `vdd`, `VdD`, `VDD`, `Vdd`, and so on. Specify the registration information in the *Regular Expression Based Registration* group box on the *Supply Nets* tab of the Power Manager Setup form.

Power Manager checks for the presence of the registered nets and identifies them as power or ground nets. These registered nets are used to create power domains.

Note: You must register the top-level supply nets for cells that do not have an associated Liberty model.

Supply NetSet Properties Prefix Registration

This includes identifying and registering of supply nets by using the property name prefix for inherited connections that have been specified in the *Supply NetSet Properties Prefix* section on the *Supply Nets* tab of the Power Manager Setup form.



For non-Liberty cells, the supply name can also be derived from the prefix of the property name used in the net expression for supply net connectivity. Power Manager creates property names for net expressions with a specific prefix as specified in the setup. Prefix matching helps to identify the power/ground type for lower level cells when the tool creates redirected netSets for resolving supply net connectivity to the top level supplies. For details, refer to Redirected netSet Property Creation and Optimization.

For power/ground type, precedence is given to supply net names that are already registered in the setup. Else, the property prefix name is used. The prefix name is registered in the setup for different supply types such as follows:

```
powerNetPropPrefix - vdd

groundNetPropPrefix - vss

pwellNetPropPrefix - hSup_sub

nwellNetPropPrefix - lSup_sub

deeppwellNetPropPrefix - hSup_sub2

deepnwellNetPropPrefix - lSup_sub2
```

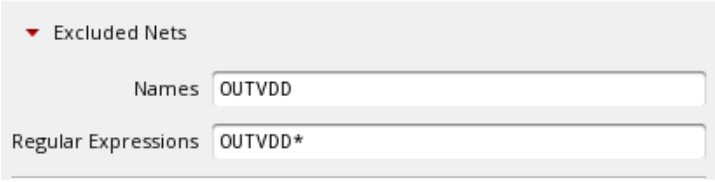
Power and Ground Nets Exclusion from Name-Based Registration

Net names with a certain pattern can be detected as a regular expression during the supply nets name-based registration. It can lead to an incorrect registration for the net as a power or ground net, for example, the Monitor and Sense pins used in the design to sense and test the supply signal level during functioning of the design.

Use the *Names* field in the *Excluded Nets* section on the *Supply Nets* tab of the Power Manager Setup form to exclude a subset of specific net names defined in the *Regular Expression Based Registration* section or defined using the signal type.

Similarly, you can use the *Regular Expressions* field in the *Excluded Nets* section to specify one or more power or ground regular expressions that have been registered using the *Regular Expression Based Registration* section or defined using the signal type. For example, if the regular expression of power net is `VDD`, the nets such as `VDD`, `VDDD`, `AVDD` `VDD_SUB`, `VDD_PH`, `VDD_PHY`, `VDD_PH1`, and so on, are specified as power nets. However, if

you want to exclude all nets containing `OUTVDDE` from being power nets, you can specify `OUTVDD` to be excluded as a regular expression.



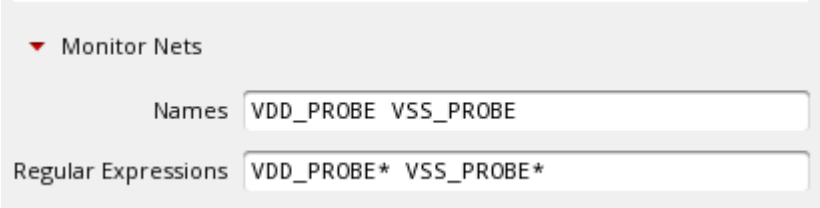
In a setup file, the excluded nets are represented as follows:

```
;;; Exclude Nets
        excludePG (nil
            names ("VDD_PROBE" "VSS_PROBE")
            regExprs( "vdd.*en vdd.*fb vss.*en vss.*fb" )
        )
```

Monitor Nets Registration

Use the *Monitor Nets* section on the *Supply Nets* tab of the Power Manager Setup form to:

- Register a net as Monitor/Sense, for specifying it as a net or port for probing or testing.
- Specify the regular expression for probing or testing.



Specify values in the *Names* field in the *Monitor Nets* section on the *Supply Nets* tab of the Power Manager Setup form to register a subset of specific net names that have been registered in the *Regular Expression Based Registration* section or defined using the signal type as Monitor Nets.

Similarly, specify values in the *Regular Expressions* field to register a subset of power or ground regular expressions that have been registered in the *Regular Expression Based Registration* section or defined using the signal type as Monitor Nets.

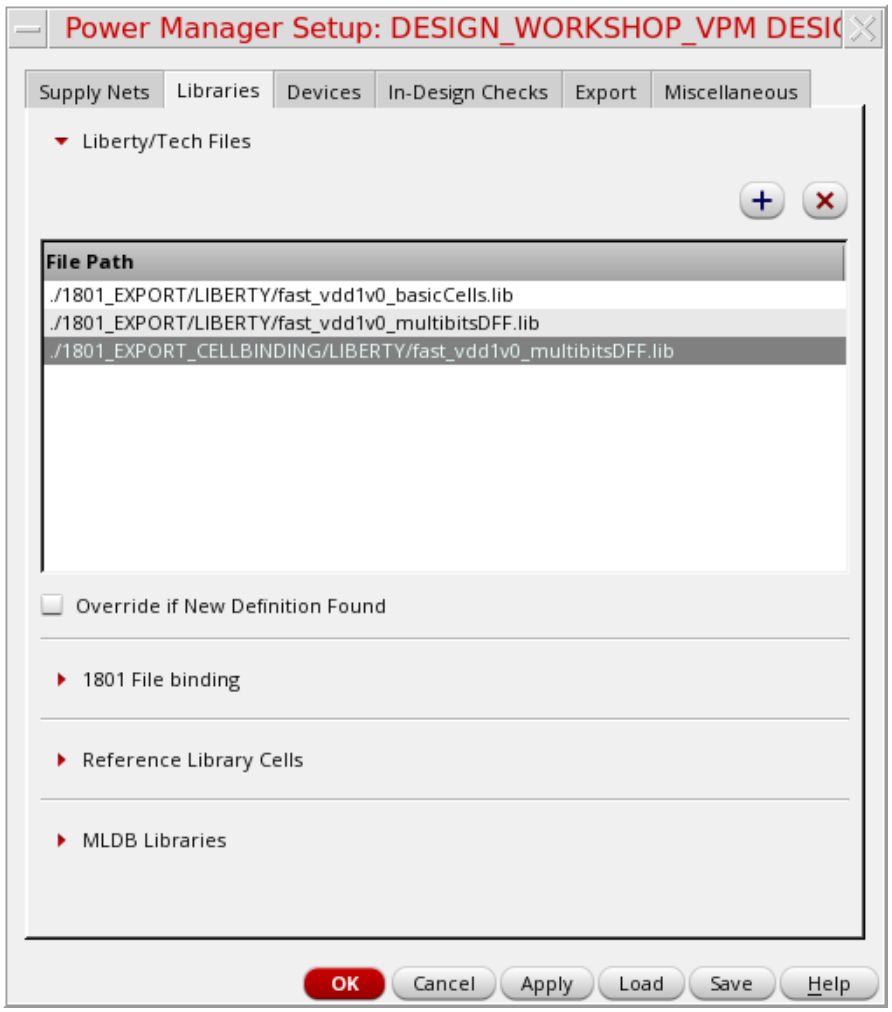
In a setup file, the monitor nets are represented as shown below:

```
        monitor (nil
            names ("VDD_PROBE" "VSS_PROBE")
```

```
            regExprs( "vdd.*sense vss.*sense" )
        )
    )
```

Library Registration

Use the registration options on the *Libraries* tab to register the libraries-related information. This information is used by Power Manager to read the standard and special cells modeling information. It is required to extract the power-related attributes of standard and special cells and avoids the need to access these cells for tracing the related supplies corresponding to the boundary ports.



Use the *Liberty/Tech Files* and *1801 File binding* sections to specify the Liberty models or 1801 special cell definition files as follows:

1. Click  to add and  to remove the Liberty/Tech Files.

Optionally, you can register a cell and bind it to its existing 1801 power intent file. The power-related information is read from the existing 1801 bound to the cell.

1. Click  to add,  to modify, and  to remove the binding information. It opens the Add 1801 File Binding form.
2. Specify the values in the *Cell* and *1801 File* fields. Click *OK*.

Note: This option is commonly used for black-boxing contents of a digital sub-block that has power intent.

Special Cell and Standard Cell Modeling

The IEEE 1801 standard is a must for the design and verification of low power integrated circuits, the correct identification and mapping of all power attributes, topology-related attributes, and the functions of low power special cells or standard cells. This is achieved by the proper registration of the special cells and the standard cell models. If cells are registered, they are identified in the schematic view accordingly.

Low power special cells and standard cells include:

- Isolation cell
- Level shifter cell
- Power switch cell
- Always on cell
- State retention cell
- Combinational/Sequential elements

Power Manager extracts level shifter strategy, isolation strategy, and power switch strategy in the power intent based on the definition provided in a 1801 special cell definition file or Liberty models. The in-design checking uses the same information to verify the power domain crossings with or without these special cell instances,

You can register these cells in any of the following three ways:

- By using the Liberty models of the cells

Special cells and standard cells, which have their Liberty model available, are recognized for their topology and various supply-related attributes by including the corresponding Liberty files. The Liberty files enable the following:

- The pg_pin attributes specified in the Liberty files help in identifying the nets used inside the cells as power or ground.

```
pg_pin(VDD)  {
voltage_name          : VDD;
pg_type               : primary_power;
direction             : inout;
}
```

- The voltage map specified in the Liberty files is used to extract the voltage levels of the corresponding supply and ground nets used.

```
voltage_map( VDD , 1.100000);
```

- The `cell` type specified in the Liberty files is used to identify the type of low power special cell or the standard cell, and extract it accordingly during 1801 power intent extraction or running In-Design Checks.

```
cell(Isolation_Cell) {
is_isolation_cell : true;
```

- The function attributes specified in the Liberty files are used as follows:
  - The switch/enable function of the low power special cells is used for finding the switch functions, enable conditions, and for backtracing the control/enable signals of the special cells, such as power switch cell, isolation cell, and enable level shifter cells, for any of the boundary ports.

```
pg_pin(VDDSW){
switch_function : "psw_en";


pin(out_iso) {
isolation_enable_condition : "en_iso";
```

  - The output function of the standard cells is used during backtracing of the control/enable signals of the low power special cells that are generated by some combinational logic, for example, output of a NAND gate. Here for a NAND gate Liberty function attribute of its output can be used.

```
pin (Y) {
direction : "output";
function : "(!((!AN) B))";
```

- By using the 1801 special cell definition file

When you provide a special cell definition file at the location specified in the setup file, before extracting the power intent, Power Manager reads the given file and recognizes the cells specified in that file as special cells. The 1801 special cell definition file aids in modeling of these cells by specifying the related attributes in their definition.

- Cell type
- Supply ports/Switchable supply ports
- Data ports
- Enable/control ports
- Voltage range
- Cell-specific attributes

Example:

```
define_isolation_cell -cells ISOHLDX1_OFF -power ExtVDD -ground VSS -
clamp_cell low -enable !ISO
define_level_shifter_cell -cells LSLHX1_TO -direction both -input_power_pin
ExtVDD -output_power_pin VDD -ground VSS -input_voltage_range {{0.8 1.4}} -
output_voltage_range {{0.8 1.4}}
define_power_switch_cell -cells HSWX1 -type header -power ExtVDD -
power_switchable VDD -ground VSS -stage_1_enable PSO -stage_1_output PSO_out -
exclude { HeadSwitch_WrapperA HeadSwitch_WrapperB }
```

Note: Use this method to register special cells for the standard and verified standard cell libraries, which do not have the Liberty models available. This method can also be used to register custom standard cells.

Setup registration for Liberty models and the special cell definition file is as follows :

```
libFiles (
"libs/fast_vdd1v0_basicCells.lib"
"libs/fast_vdd1v2_extvdd1v0.lib"
"libs/macro_lib.lib"
"libs/Headswitch.upf"
)
```

Design Sub-Block Modeling During Top-Level 1801 Design Model Extraction

The modeling information of hierarchical design blocks can be provided to digital verification tools, such as CLP as follows:

- Liberty macro models: The contents of analog sub-blocks need to be modeled in terms of Liberty power model export to be provided to CLP. For details, see Exporting Power

Intent of a Design. CLP identifies the modeling information and reads the power-related attributes of the boundary ports of the block for crossing analysis.

The Macro Liberty power model for a sub-block can be specified as a Liberty file on the *Library* tab of the Power Manager Setup form and therefore, binding it to that block.

- 1801 file binding - If the power intent of a hierarchical sub-block is already available as a 1801 design model, the same can be bound to that block in the *1801 File binding* section on the *Library* tab of the Power Manager Setup form. These blocks are locked for extraction of 1801 power intent by Power Manager. The tool does not traverse down the hierarchy but just one level inside its schematic to make the interface connectivity to the top-level 1801 file. A 1801 file used for importing the power intent on a digital sub-block can also be used for cell binding of that block.

  During the top level 1801 design model extraction, Macro Liberty Power Model, the special cell definition file and the available power intent of the sub-blocks are consumed together. The top-level design model can further be verified using CLP.

  Cell bindings can be registered in the setup as:

  ```
  cell1801Bindings (
  ("data_LevelShifter" "./data_LevelShifter.upf")
  )
  ```

  Note: If the top design is large, the power intent extraction step might lead to memory-related issues. Therefore, for large designs, follow the bottom-up approach. In this approach, you can first extract the power intent of sub blocks separately in 1801 files and then associate the 1801 files with their respective instances in the top design. After this, when you extract power intent for the top design, the tool consumes less memory. Also, you can apply the bottom-up approach when extraction options cannot be shared between the top design and the sub blocks. For example, when the power / ground nets definitions, power / ground net voltage definitions, or power net-ground net voltage pairs definitions in the top design are different from the sub block definitions.

Reference Libraries or Cells

If the design contains instances of any library or a particular cell, apart from the Liberty cell, it can be read-only, black box, or a reusable block. You can register those reference libraries. The cells in reference libraries are not edited during the 1801 import flow. If the input 1801 file has a connect supply net command for a instance inside the blackbox cell, the cell is not edited to create the netSet or any supply ports. During import, the hierarchy is traversed to map all the supply nets to the supply set at the instance boundary.

Reference library registration in the setup is as follows:

```
refLibCells(
```

```
("analogLib" "*buf*")
```

MLDB Libraries

Power Manager processes the Liberty files to read the standard cell, special cell, and macro cell model definitions and stores them in an OA-based proprietary database. This avoids the need to reprocess the Liberty files every time a command is executed. Such a database is called Model Library database (MLDB). MLDB reads and stores the PG pins, pins, and the non-characterized attributes for the various cell models provided through Liberty files. These attributes are saved in a different library than the design or PDK library, which can be utilized in various projects to extract details instead of parsing the Liberty files every time. This can save a lot of run time that is required to parse large Liberty files if it has to be done repeatedly for extracting power-related attributes. If no user-specified library is provided, the database is in-memory and can be used only for a particular Virtuoso session.

MLDB creation requires a new library to be created. The library includes a `tech.db` file, which contains all the power-related attribute information that is extracted by a single traversal from the registered Liberty files. The setup registration for MLDB is:

```
projectMldbLibName "MLDB_TESTCELL"
```

A new `.lib` file is added to MLDB only in one of the following conditions:

- You have re-imported a modified setup specifying additional `.lib` files and removed `.lib` files that are not required anymore.
- You have run a VPM command, such as extract power intent or import power intent.

The modified MLDB is saved when closing all the files or exiting Virtuoso.

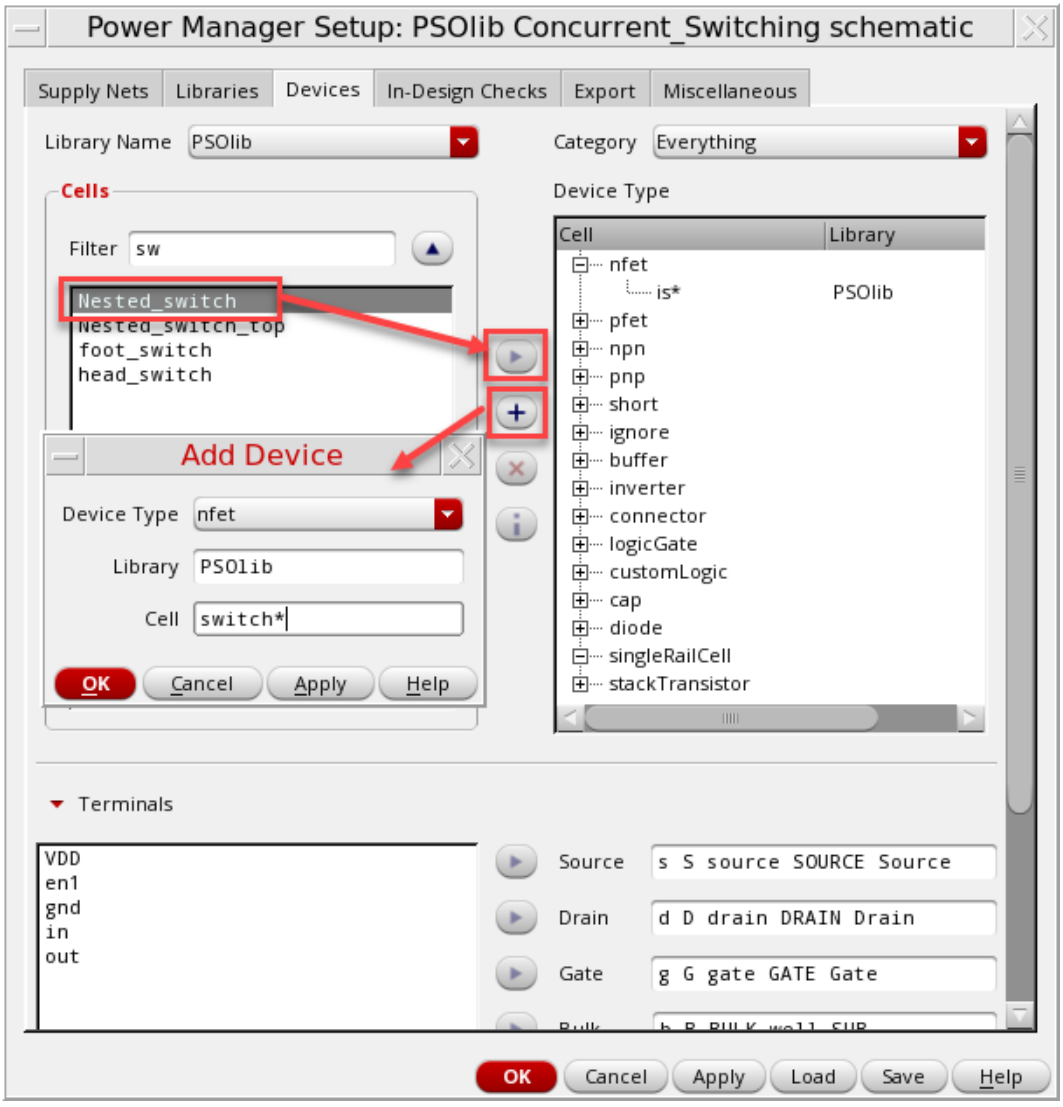
Device and Cell Registration

Register the device and terminal related information on the *Devices* tab of the Power Manager Setup form. This information is used by Power Manager to read the device type and its topology in the design. For devices with more than two terminals, you can associate terminals to have specific mapping. Correct recognition and identification of device type is important for the supply traversal to find the related supplies of the boundary ports associated with the design logic.

To register cells as devices:

1. Use *Filter* in the *Cells* group box to search for a cell. The *Filter* option gives the flexibility to easily filter and sort the desired choice of devices to be registered.
2. Select a device or a set of devices to register.

3. Select the device type for assigning the devices.
4. Use  to perform the assignment.
5. Click  to remove the assignment.
6. Specify conditions/criteria with the assigned device by clicking  .



The devices can also be registered using a wild character based registration by using the Add Device form, which is opened by clicking  . Specify a library and cell name based on a wild character notation and register the device in the corresponding Device Type. For example, specifying `gpdk*` and `nmos*` registers all the devices with a name matching the pattern `nmos*` in all the libraries with the name matching the pattern `gpdk*` to the device type `nfet`.

If the Library Name or a cell name field is left blank, all available libraries (for a particular cell name pattern) or all available cells (for a particular library name pattern) will be accounted for and register based on the wild character specification.

Devices used in the design, such as MOS devices, resistors, capacitors, and diodes, need to be registered in the setup under the following categories.

- Transistors (nfet, pfet, npn, pnp)
- Short devices (including devices with n terminals)
- Pad cells
- Cells to be ignored

In a setup file, the device registration is represented as shown below:

```
;;; Devices
    devices (nil
        pfet (
            (nil "pmos*" nil)
            ("analogLib" "pmos*" nil)
            ("gpdk*" "pmos*" nil)
        )
        nfet (
            (nil "nmos*" nil)
            ("analogLib" "nmos*" nil)
            ("gpdk*" "nmos*" nil)
        )
        pnp (
            (nil "pnp" nil)
            ("analogLib" "pnp*" nil)
            ("gpdk*" "pnp*" nil)
        )
        npn (
            (nil "npn" nil)
            ("analogLib" "npn*" nil)
            ("gpdk*" "npn*" nil)
        )
```

Passive Devices

Passive devices, such as resistors, capacitors, and diodes need special handling and registration in the setup as per their usage in the design.

Short Devices

Resistors are registered in setup as short devices. You can short the terminals of devices that have more than two terminals.

In a setup file, the short device registration is represented as shown below:

```
short (
("analogLib" "res" nil)
("myLib" "wireshortcell" nil (nil shortedTerminalMap (("t1" "t2")("t3" "t4" "t5")
)))
)
```

Ensure to register passive devices associated to a condition as mentioned in the setup file. For example, certain configurations in designs might require the resistors not to be registered as short devices.

The multi-term shorting of devices under the Short category is considered only during macro model extraction. However, it is not considered while netlisting, which is a part of CLP integration. Therefore, it leads to an error during CLP verification.

Connectors

You can register cells under the connectors in the following scenarios:

- While relating power nets with ground nets for establishing power ground net pairing. When the extractor traces the power to the ground paths in order to pair power and ground nets, if there is a connector instance in the path, the related terminals of the registered connector instance (a multi-terminal cell) help to continue tracing from one end to the other end.
- While backtracing the data path from the control pins of the special cells to the macro or design ports to generate the enable or shutoff condition for low power rules or power domains.

In a setup file, the connector registration is represented as shown below:

```
connector (
    (nil "stdBuf*" nil)
    ("ALIB" "PMU" nil (nil shortedTerminalMap (("p" "q") ("o" "y"))))
```

```
        )
```

Capacitors used in the data path mostly for the purpose of AC coupling are treated as connectors. The capacitors that are directly connected to power or ground for the purpose of blocking DC signal are treated as ignored or open. These are blocked for supply/ground tracing. The multi-terminal capacitors are also registered as devices.

In a setup file, the capacitor registration is represented as shown below:

```
    cap (
            (nil "cap" nil)
            (nil "mimcap" nil
                (nil
                    shortedTerminalMap  (("PLUS" "MINUS"))
                )
            )
               )
```

Note: You can also use the wildcard character in the Ignore, Short, Buffer, and Inverter registration.

Diodes can be used in the design for the different purpose like in an antenna or ESD clamp structures. Their registration in a setup is specific to the polarity configuration in which they are connected.

In a setup file, the diode registration is represented as shown below:

```
diode (
("analogLib" "diode" nil (nil pTerm "PLUS" nTerm "MINUS"))
)
```

A diode is referred to as an antenna diode if the `nTerm` of diode is connected to power and `pTerm` of diode is connected to a pin. Or, `pTerm` of diode is connected to ground and `nTerm` of diode is connected to a pin.

Antenna diode-specific attributes are printed for input & inout terms connected to the antenna diode.

Power Type:

```
pin (MyInput) {
antenna_diode_related_power_pins : "VDD";
```

```
...
}
```

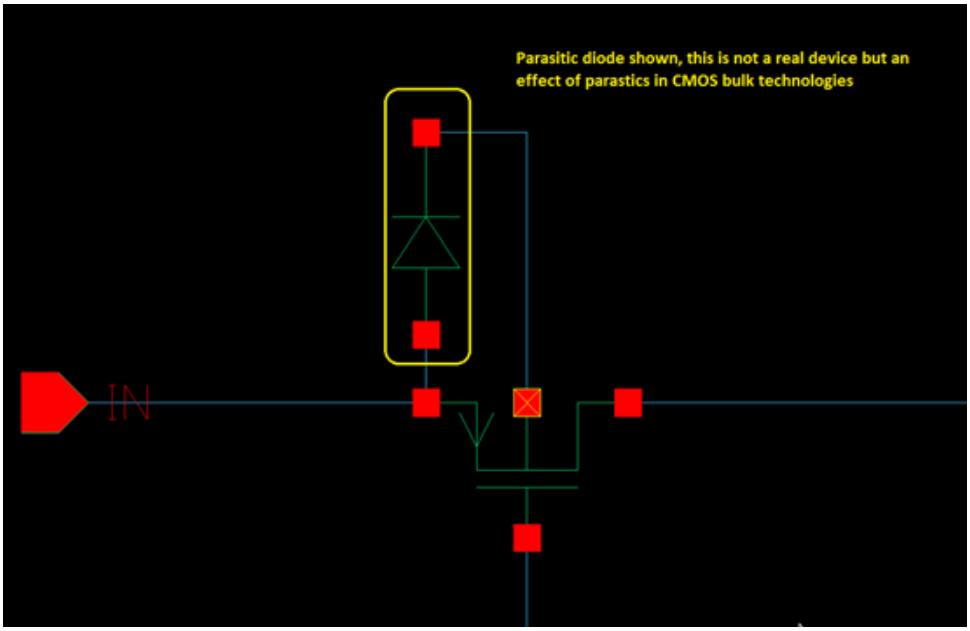
Ground Type:

```
pin (MyInput) {
antenna_diode_related_ground_pins : "VSS";
...
}
```

Power and Ground:

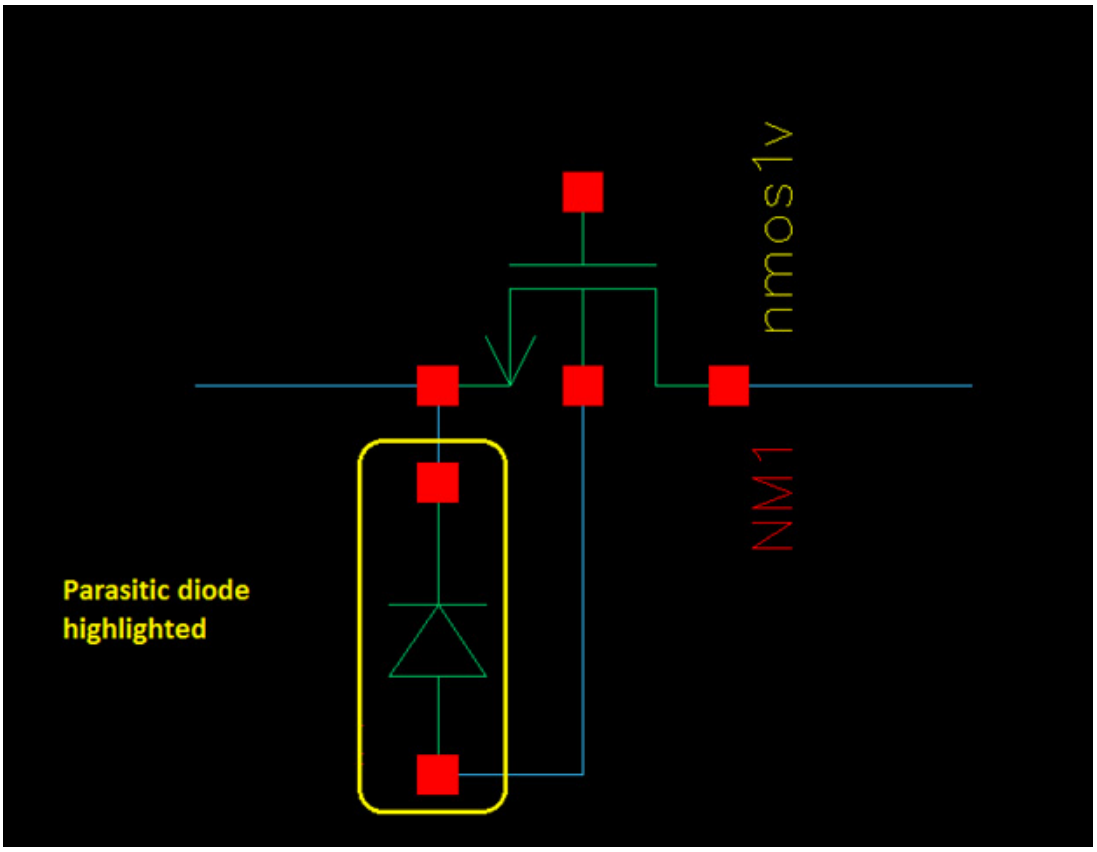
```
pin (MyInput) {
antenna_diode_related_power_pins : "VDD";
antenna_diode_related_ground_pins : "VSS";
...
}
```

A parasitic diode is formed between the source or drain terminal of a MOS device and the bulk terminal. Here is an example of a PMOS device and a parasitic diode.



For a PMOS device, a diode that has the anode connected to the signal net and the cathode connected to the power supply is extracted. In this case, it is a power type diode.

```
antenna_diode_related_power_pins : "VDD";
```



For an NMOS device, a diode that has the cathode connected to signal net and the anode connected to ground supply is extracted. In this case, it is a ground type diode.

```
antenna_diode_related_power_pins : "VSS";
```

Backtracing Enable Signals of Special Cells

This involves backtracing of the internal net to map the enable or control signals of special cells to the boundary ports. Backtracing is required to get the appropriate functional attribute related to the enable or control signals in the Liberty macro model after detecting the corresponding function of the logic generating the signals.

Liberty registration for standard and special cells in the setup is a mandatory requirement to extract a correct function for the enable or control signals of low power special cells as per the design implementation

The backtracing is supported for following registered configurations:

- For buffers, shorts, and inverters, the two-terminal devices with single input and output, such as inverters and buffers, can be backtraced. The polarity of the signal does not change while backtracing through instances, unless the instances are inverter cells. You can register these cells as Inverter. If an instance with more than two terminals is present in the path, backtracing stops and an error message is displayed. Inverter and buffer registration is done as shown below.

```
inverter (
(nil "stdInv*" nil)
("gpdk*" "inv*" nil)
)

buffer (
(nil "stdBuf*" nil)
("gpdk*" "buf*" nil (nil shortedTerminalMap (("p" "q") ("o" "y"))))
)
```

  Note: The terminal map is a mandatory requirement for backtracing when a cell contains multiple input and output and is registered as an inverter or a buffer.

- For standard isolation or level-shifter instances, signals are backtraced from output pin to input pins. For power switch instances, the signals are backtraced from the acknowledge pin (buffered enable) to enable the pin of the instance. This lets the backtracing to be performed for the control signals of all the power switch cells to the same macro port resulting in the correct enable and shutoff conditions.
- For simple logic gate, the backtracing is done through the special cell enable pin to the output pin of the logic gate that has multiple input pins is supported. All input pins of a logic gate can be backtraced separately to the maximum extent possible.

However, a cell can be registered as LogicGate (simple logic gate) only if it matches any of the following criteria:

- There are no multiple output pins in the cell.
- The boolean function is present in the setup file.
- Only AND, NAND, OR, NOR, and XOR Boolean functions are considered.

The cells are registered as LogicGate devices along with Boolean functions of the cells.

```
logicGate (
("sample" "or*" nil (nil booleanFunction "OR"))
("sample" "xor*" nil (nil booleanFunction "XOR"))
("sample" "and*" nil (nil booleanFunction "AND"))
("sample" "nand*" nil (nil booleanFunction "NAND"))
("sample" "inv" nil (nil booleanFunction "NOR"))
```

```
)
```

A custom logic can also be registered with the desired terminal mapping between its output and input ports to be suitably backtraced if it lies in between the enable/control signal and the boundary port generating it.

```
        customLogic (
            ("ALIB" "WiFi" nil (nil relatedTerminalMap
"y1=x1&x2,y2=x3|x4,y3=x5^x6"))
        )
    )
```

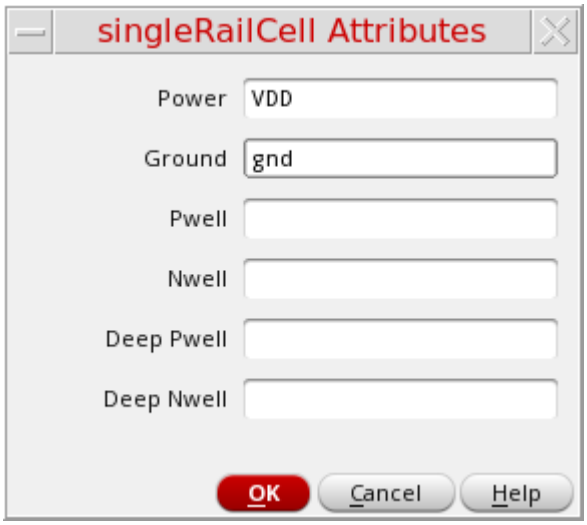
Single Rail Cell

Single rail cells are custom cells. These can be custom standard cells, an unimplemented subblock, or a complex hierarchical custom logic having a single primary power and ground supplies. Usually, a Liberty model does not exists for such cells. Consequently, the single rail cell registration enables blackboxing of any hierarchical subblock so that the extractor recognizes it as a leaf cell. All ports connected to a single rail cell (directly or through some elements) are mapped according to the port definitions as defined in standard cell registration.

Single rail devices commonly used have the following features:

- Single rail device registration can be done with global attributes set at the cell level for multiple cells. In addition, the registration can also be done explicitly for a single cell.
- Defining the global supply values is optional.
- By default, all non-supply and non-bias pins are considered to be data pins. These data pins are related to the supply pins (RPP/RPG) and the bias pins (RBP).
- Single rail cells are also supported for bias supplies, which might be different from primary supplies.

On the Device tab in the Power Manager Setup form, when you click after selecting the `singleRailCell` device type, the singleRailCell Attributes form opens. Additionally, you can view the form by selecting any cell registered as a single rail cell device.



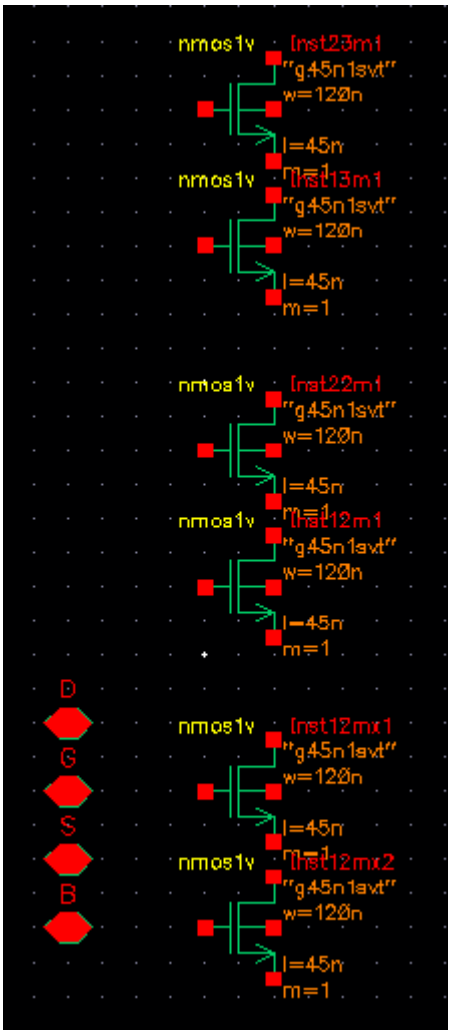
In a setup file, the singleRailCell registration is represented as shown below:

```
singleRailCell ( nil
                 cells (
                     ("myLib"    "aBuf*" nil)
                     ( nil              "bBuf*" nil)
                     ("myLib"    "cBuf*" nil
                         (nil
                             power                      "VDD"
                             ground                    "VSS"
                             pwell                       "VPP"
                             nwell                       "VNN"
                             deeppwell             "VPPP"
                             deepnwell             "VNNN"
)
                         )
                 )
                 power          "VDD"
                 ground         "VSS"
                 pwell             "VPP"
                  nwell           "VNN"
                 deeppwell   "VPPP"
                 deepnwell   "VNNN"
)
```

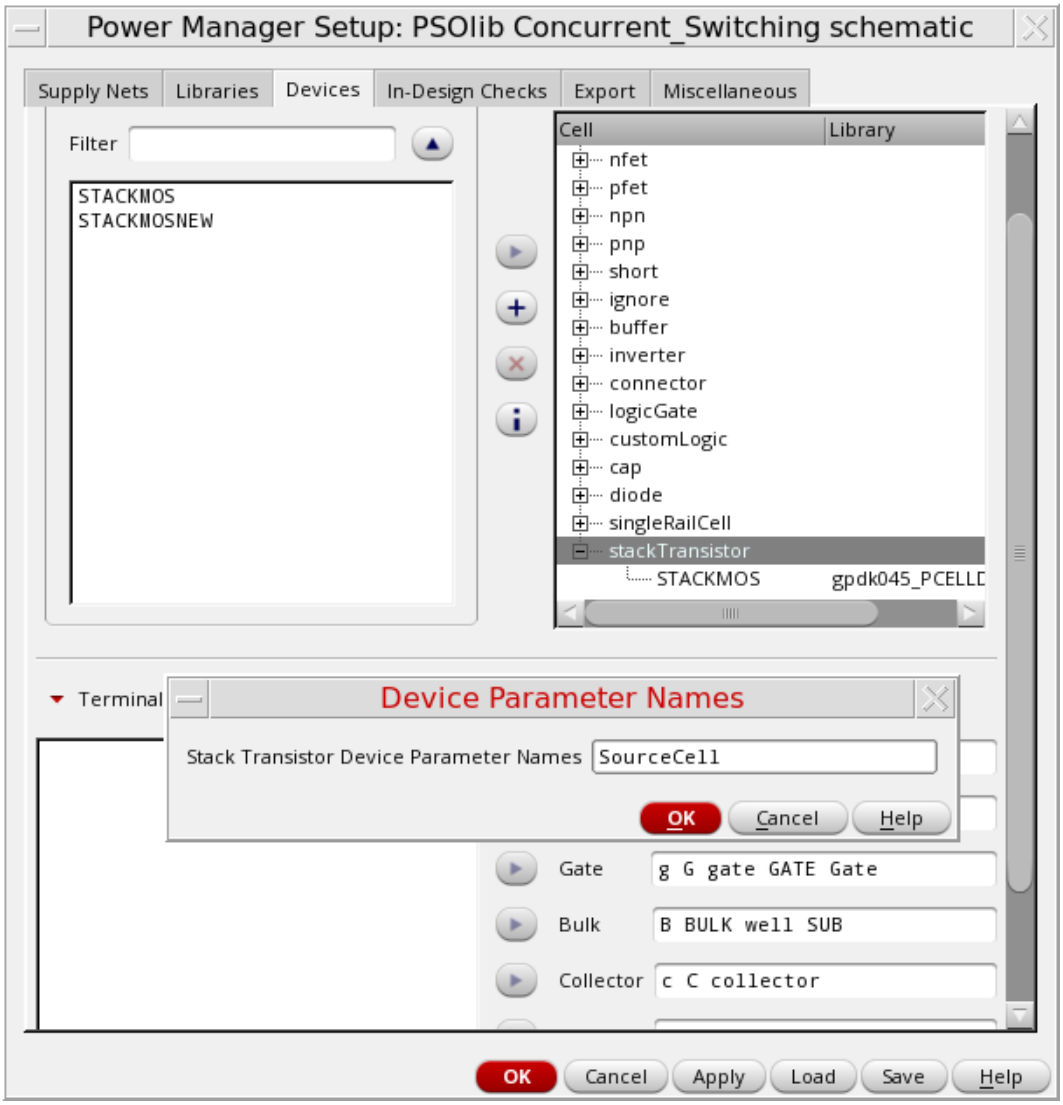
Stack Transistor

Stack Transistor devices are used in the advanced node designs for reduced power consumption at lower gate lengths. Stack Transistor devices commonly used have the following features:

- The same master is used for creating different device variants, such as PMOS or NMOS.
- The Stack Transistor device has a series of cascaded PMOS or NMOS devices inside.
- A parameter on the device Pcell master defines the device type used inside for stacking.



On the Device tab in the Power Manager Setup form, when you click  after selecting the `stackTransistor` device type, the Device Parameter Names form opens.



When you click the `stackMosfet` device type, the Device Parameter Name form opens.

It is used to specify the device parameter names for the stack MOS device to define the device as PMOS or NMOS. You can specify more than one parameter names.

Note: If the device terminal have names different from the conventional naming, you need to provide a terminal mapping for the same during device registration.

In a setup file, the stack MOS device registration is represented as shown below:

```
stackTransistor (
```

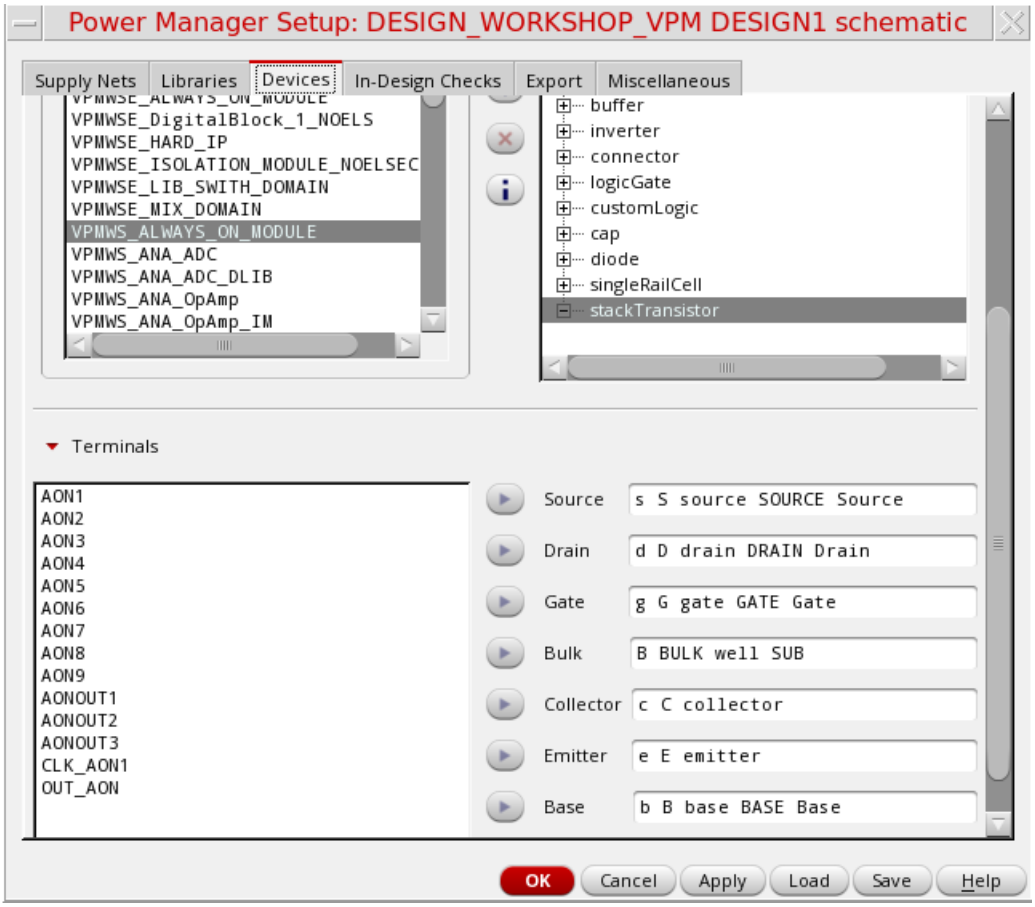
```
(nil "STACKMOS*" nil)
("analogLib" "STACKMOS*" nil)
("gpdk*" "STACKMOS*" nil)
)
stackTransistorDeviceParamNames "SourceCell"
```

Terminal Name Registration for Devices

While traversing through the connections of transistor instances, the Power Manager needs to identify the gate, source, and drain terminals of the registered transistor devices. Therefore, in addition to registering the devices, you need to register the names of the device terminals of these devices. You can register terminal names with an appropriate terminal type in the Power Manager Setup form and subsequently, in the setup file as shown below:

```
txTermTypeNames (nil
source      "s S source SOURCE Source"
drain       "d D drain DRAIN Drain"
gate        "g G gate GATE Gate"
bulk        "b B BULK well SUB"
collector   "c C collector"
emitter     "e E emitter"
base        "b B base BASE Base"
substrate   "S BULK well SUB"
```

)



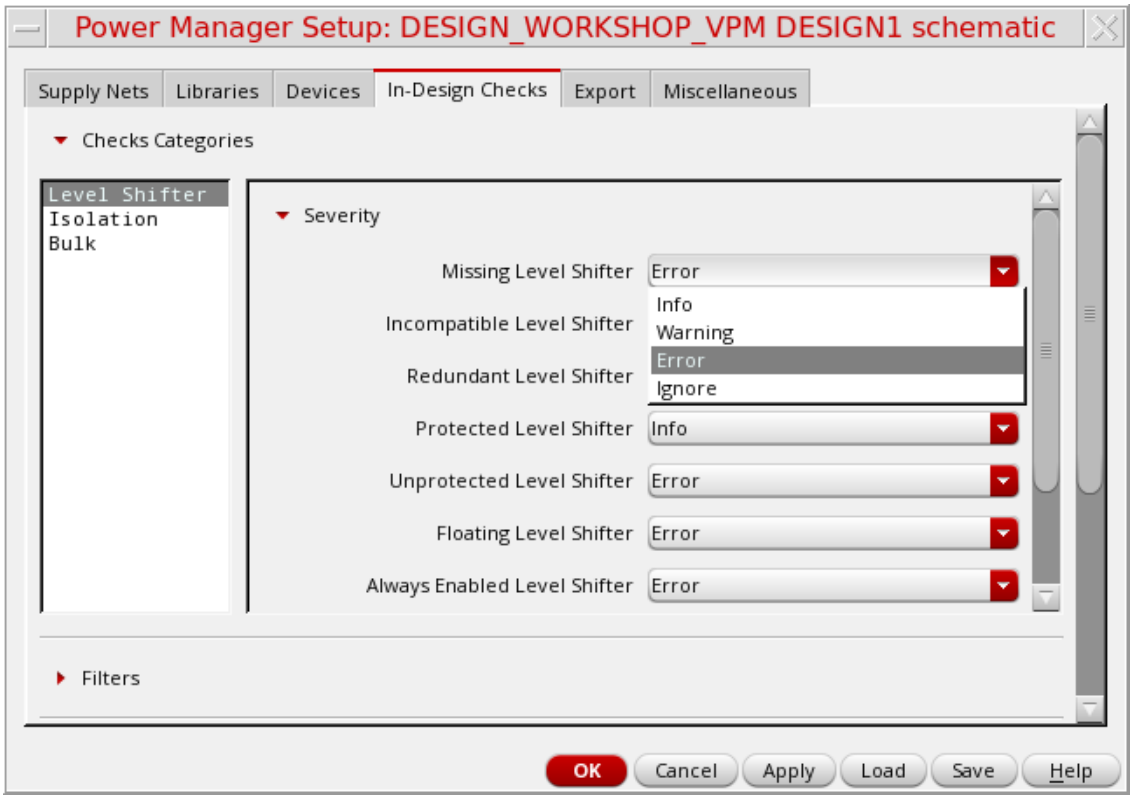
Caution

If you are using a PDK that is not provided by Cadence, it is essential to register the transistors and terminal names to ensure correct extraction of boundary ports and automatic pairing of power nets and ground nets.

In-Design Checks

Power Manager supports running a set of predefined and configurable circuit checks for mixed-signal designs that include devices and standard cells. You can define the severity of reporting of these checks on the *In-Design Checks* tab.

In the Checks Categories section, specify the severity level of the results reported for the *Level Shifter*, *Isolation*, and *Bulk* checks.



Note: For reporting violations generated from a specific check, the error severity for that check must be set to `Error` or `Warning`. If set to `Ignore`, the reporting of violations for the corresponding check is disabled.

The *Tolerance* section provides the flexibility to add tolerance for power and ground rail supply net voltage values for level shifter checks. All data path crossings within the tolerance limits are not flagged as violations. All the upper bound tolerances are positive numbers or zero and the lower bound tolerances are negative numbers or zero. The missing level shifters are reported for the signal crossings operating at different voltages when at least one of the following conditions, indicated by the GUI options, is true:

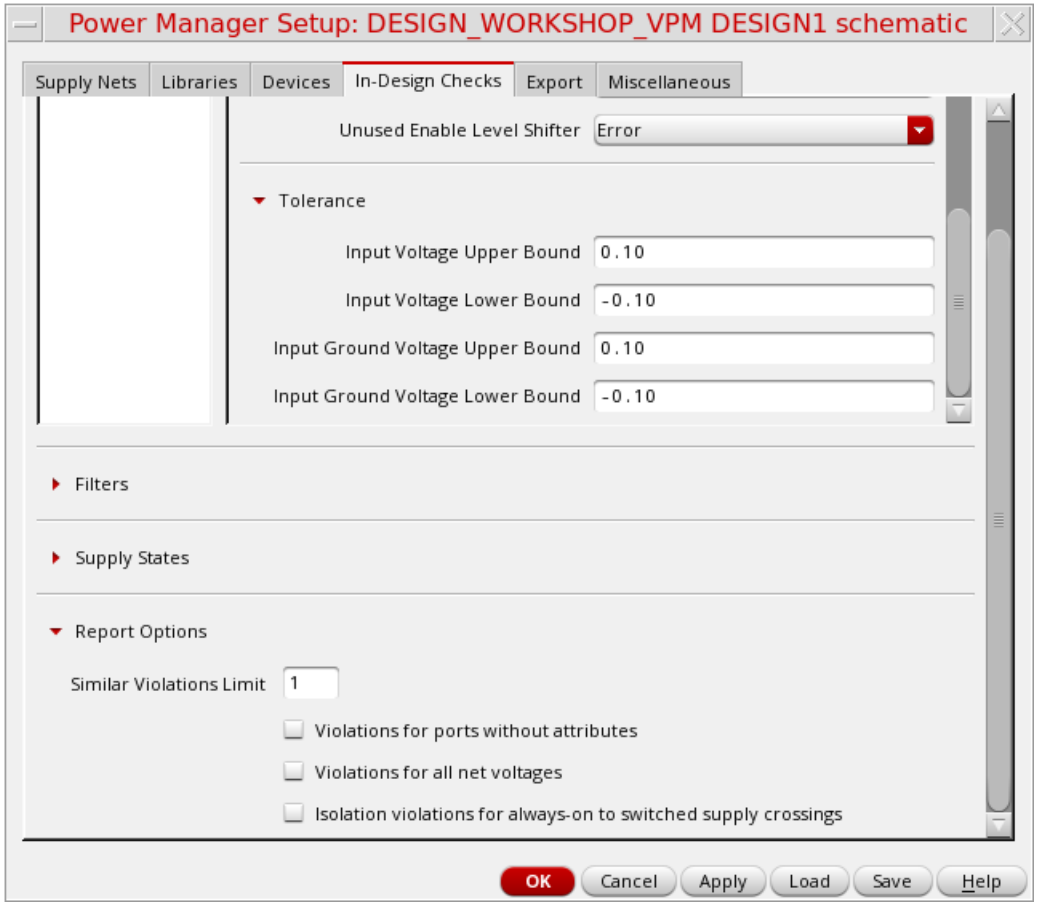
- *Input Voltage Upper Bound*: The driver power voltage is more than the receiver power voltage by the upper bound input voltage tolerance. This value should be >=`0`.
- *Input Voltage Lower Bound*: The driver power voltage is less than the receiver power voltage by the lower bound input voltage tolerance. This value should be <=`0`.

- *Input Ground Voltage Upper Bound*: The driver ground voltage is more than the receiver ground voltage by the upper bound ground input voltage tolerance. This value should be >=0.

  The following table shows an example of scenario's which would reflect a level shifter error and the ones which would not reflect the error.

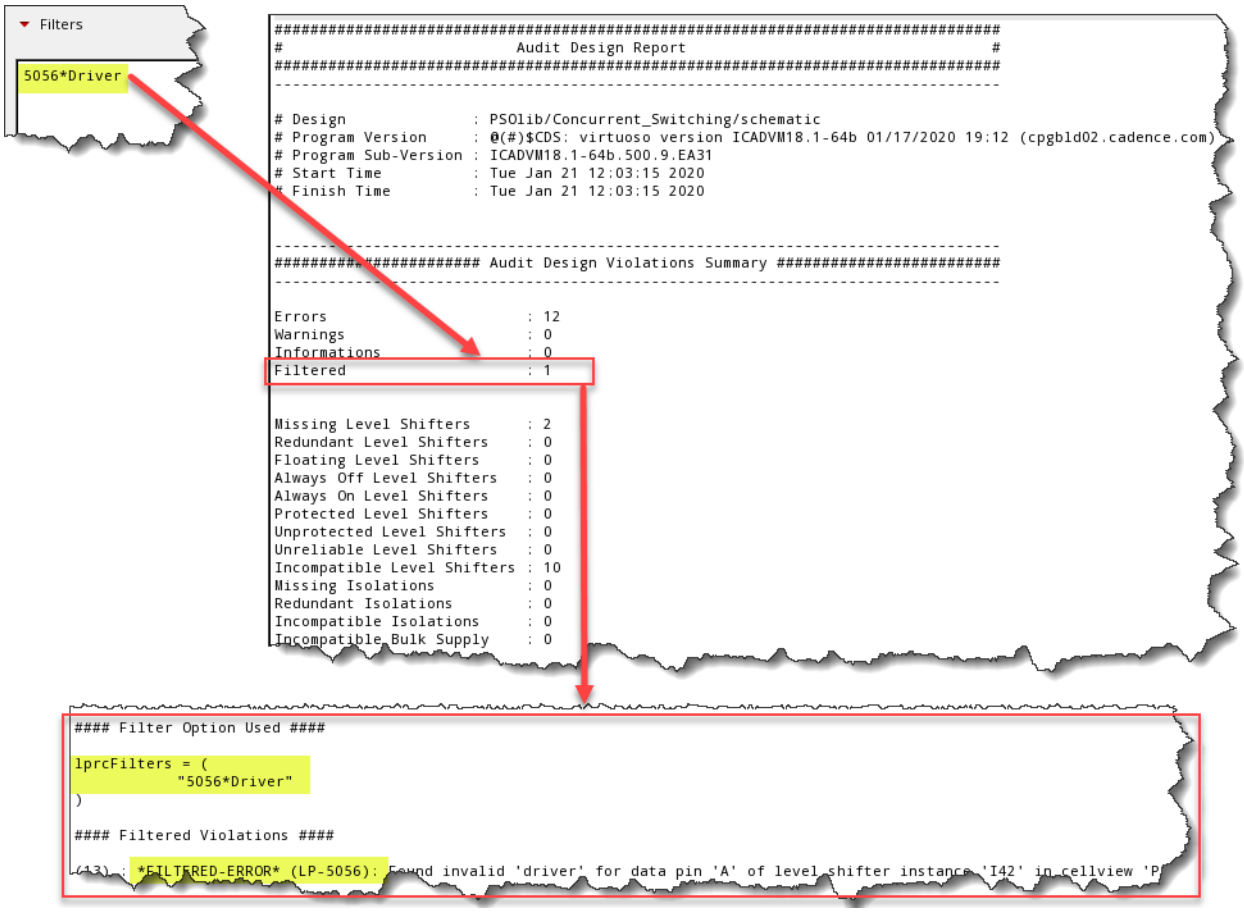
VDD1 (Driver)	VDD2 (Receiver)	Diff	Upper Bound	Lower Bound	Error Status
1.4	1.6	-0.2	0.1	-0.1	Error
1.4	1.6	-0.2	0.1	-0.3	No Error
1.5	1.6	-0.1	0.1	-0.1	No Error

- *Input Ground Voltage Lower Bound*: The driver ground voltage is less than the receiver ground voltage by the lower bound ground input voltage tolerance. This value should be <=0.



You can add filter patterns in the *Filters* field to filter some of the error and warning messages.

These errors are reported and exist in the design, but these are not a problem for the design in the given scenario.



When a design uses multiple voltages for efficient power management, they can be switched on and off or scaled up or down to different voltage values as per design requirements. These are power states. A power state can be defined as a set of power and ground nets with their respective voltage values. IC designers choose the states as per the performance need of a design. Some common schemes that use the power states are dynamic voltage, frequency scaling, and run and standby performance modes.

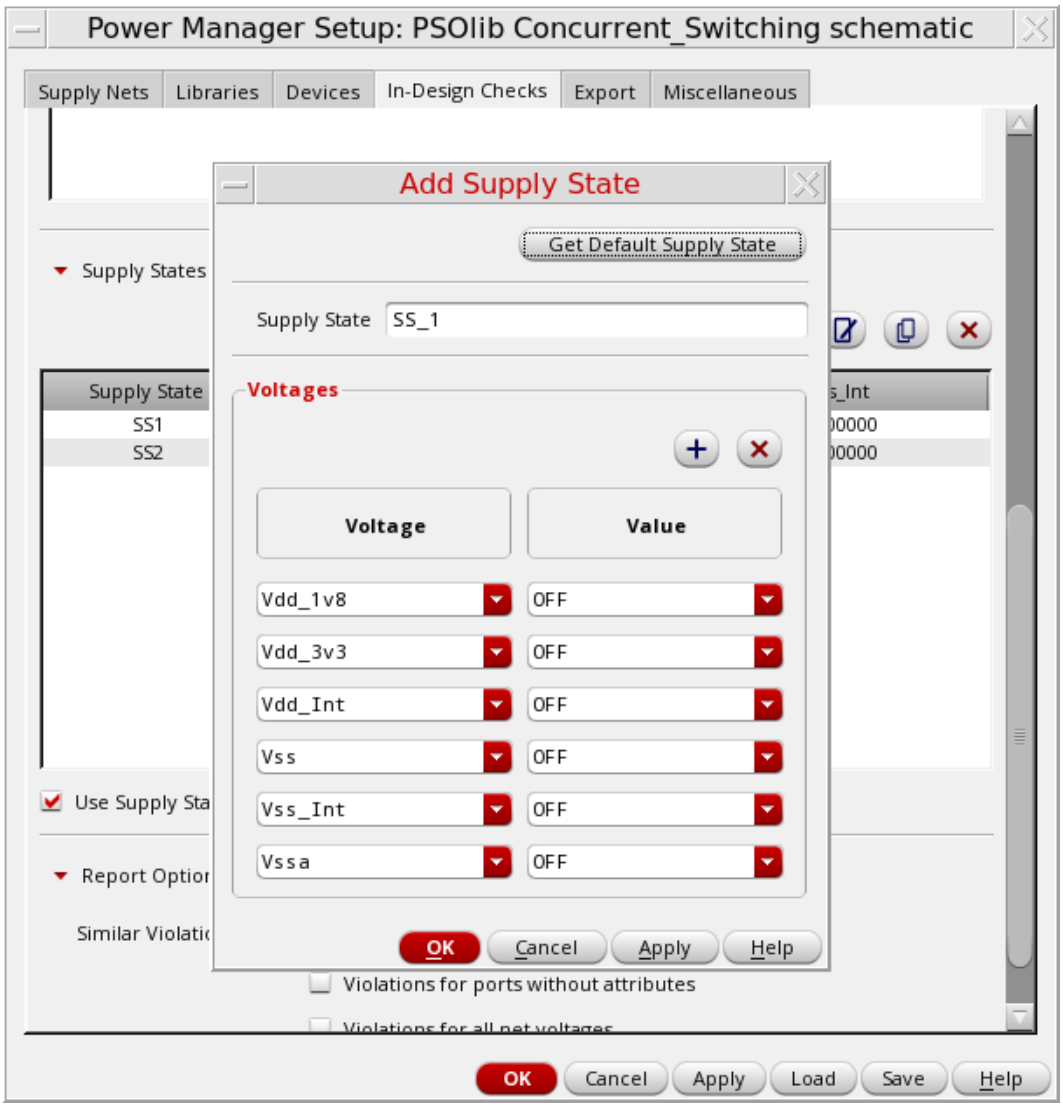
As the number of power and ground supplies increases, their combination increases multiple folds. To perform checks, it is critical to define the valid power states. Check the specified supply states in the design. A voltage net can be ON at different supply voltages and off at a particular voltage value.

For a power net, OFF is considered as 0 and ON can be 1.0V, 1.1V, 0.8V, and so on.

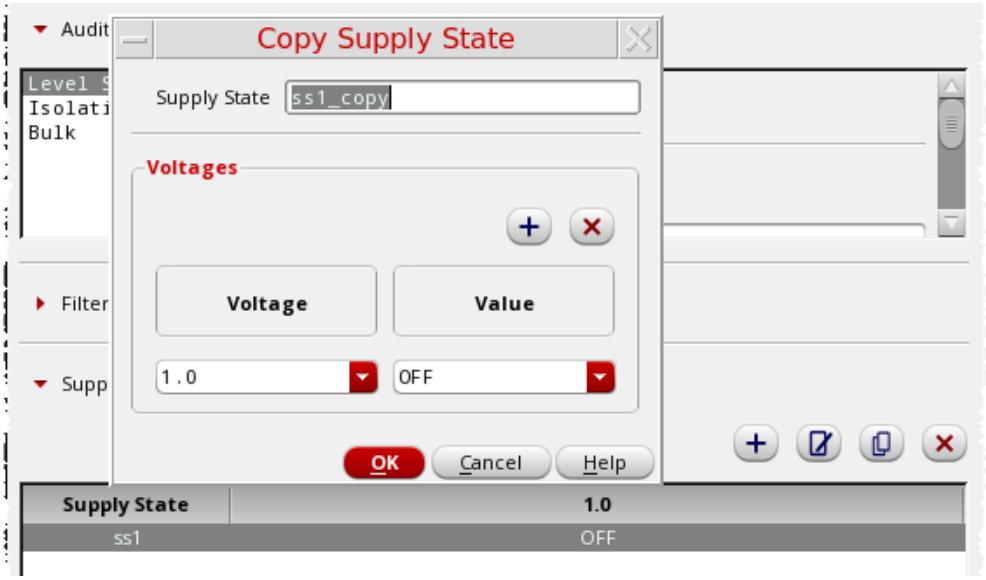
For a ground net, OFF is considered as the signal that is not available.

To populate the supply states:

1. Click + to add the supply states in a design. It opens the Add Supply State form.
2. Click the *Get Default Supply State* button to remove all the existing voltages/values pairs from the Add Supply State form if already existing.You have an option to apply this default state directly or after adding/deleting/modifying some supplies and their values. The default supply states have a unique non existent name.
3. Optionally, click ⧉ to modify, and × to remove the supply states.

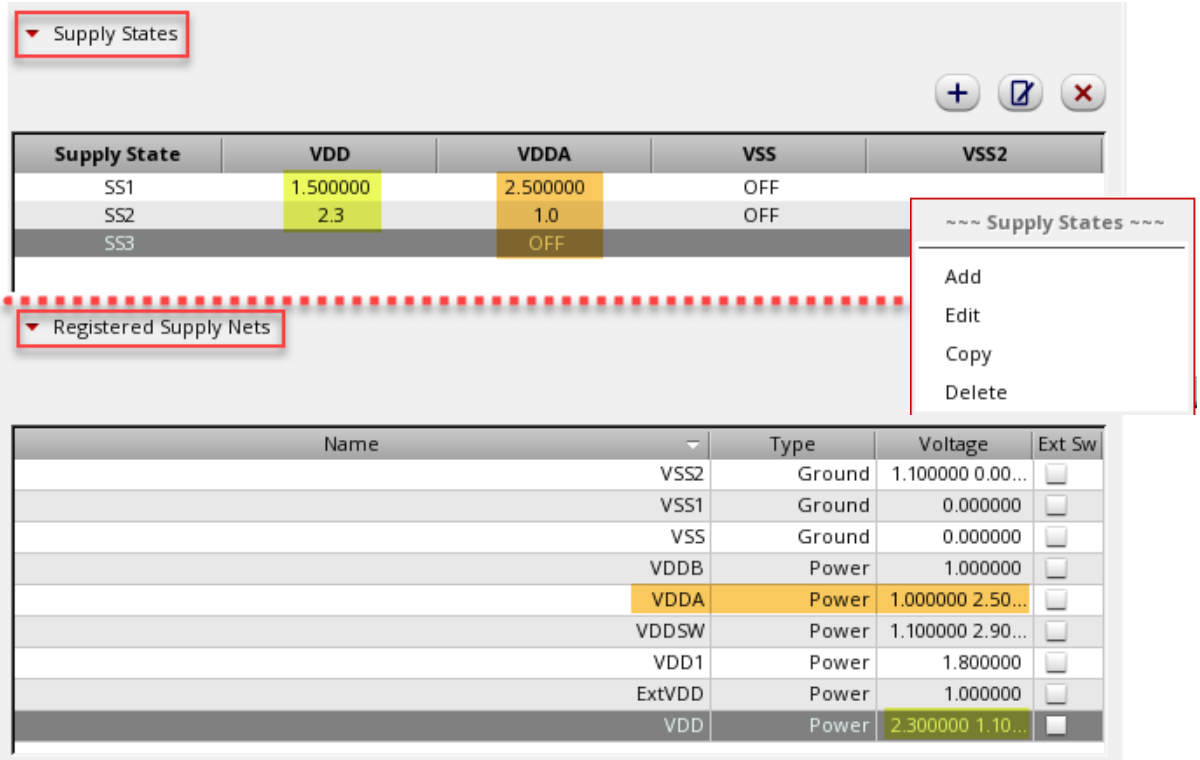


4. Additionally, use  to copy existing supply states from the table. The Copy Supply State form shows all the supplies with their respective voltage values of the currently selected *Supply State* in the table. By default, the name of the copied state is shown as "State Name + _copy". You can add, delete, or edit voltages and their values. You can also modify the name of the state.



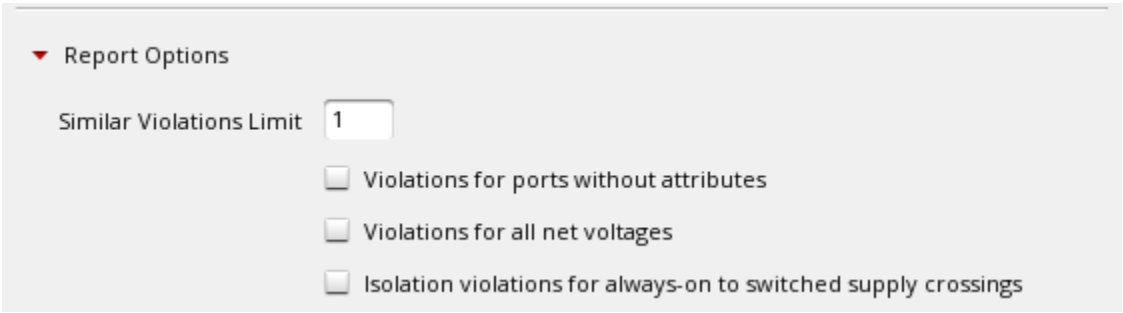
The *Registered Supply Nets* table in the supply nets tab is always in synchronized with the Supply States. If the supply nets are already available and registered in the *Registered Supply Nets* table, the same nets are available for selection in the Add Supply State form along with different voltage values in the lists. Conversely, while filling the Add Supply State

form, if you introduce a new supply net with a set of voltage values, the values are automatically updated in the *Register Supply Nets* table.



Note: The supply information registered in the *Supply States* table is read by the Power Manager when the *Use Supply States* option is selected. If it is not selected, the supply information is read from the *Registered Supply Nets* table in the Supply Nets section of the Power Manager Setup form.

Select the check boxes in the *Report Options* section based on your preferences.



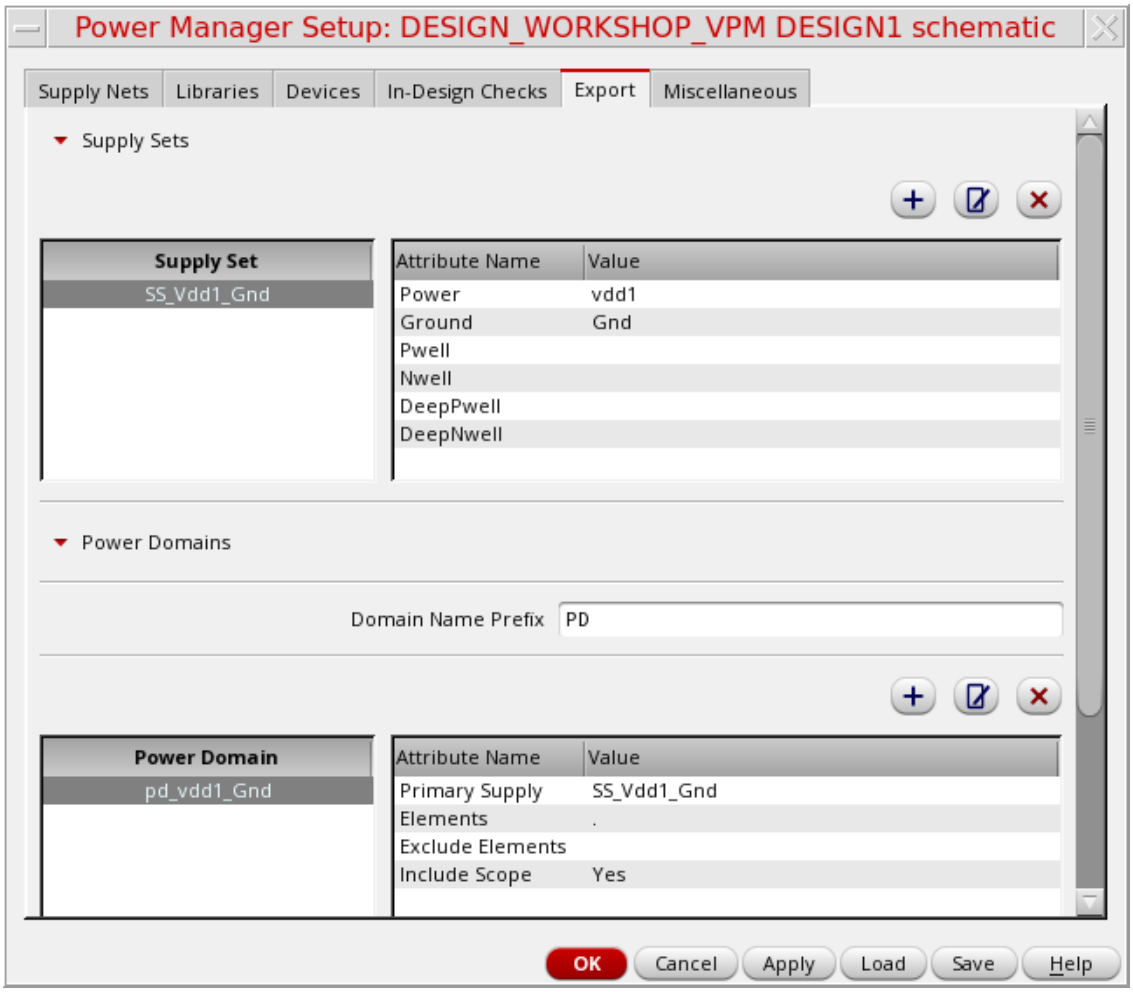
- *Violations for ports without attributes*: Performs checks for the boundary ports that do not have any associated attributes, such as driver or receiver supply sets. By default, in-design checks do not report violations at the boundary ports unless information about the related power and ground nets of the boundary ports is specified using the registration information. For details, see Port Attributes Registration.
- *Similar Violations Limit*: Defines the maximum number of violations that are similar and should be reported. The default value is `1`. If the value is `0`, all similar violations are reported.
- *Violations for all net voltages*: Reports violations for all net voltages specified in various supply states or different voltage values registered in the *Registered Supply Nets* table provided in the setup.
- *Isolation violations for always-on to switched supply crossings*: Reports violations for a missing isolation cell at the always-on to switched supply crossings.

Supply Set and Power Domain Registration

A power domain is a part of the design that operates at a specific voltage. While capturing the power intent, you need to specify the power domains for the top level of the design. There can be a single or multiple power domains associated with the different sections of the design addressed by a specific supply set.

During automatic extraction, Power Manager identifies a power net and a ground net from the pair defined explicitly in the setup as a supply set in the *Export* tab of the Power Manager Setup form. It associates the supply set with the power domain as specified in the setup. You can specify a supply set as a primary supply set for the power domains mentioned in the

setup. In addition, you can also define the supply set for the power and ground that cannot be paired by traversing the design hierarchy or parsing the Liberty file.



Click  to add,  to modify, and  to remove the supply set and power domain information. It opens the Add Supply Set and Add Power Domain forms, respectively.

Important

The Supply Set field is automatically populated if the field is found blank or the user-defined names are in the `SS<delimiter>XYZ<delimiter>ABC` format. The automatically generated supply set names follow the `<SS><delimiter><powerSupply><delimiter><groundSupply>` format.

Important

The Power Domain field is automatically populated if the field is found blank or the user-defined names are in the `PD<delimiter>XYZ` format. The automatically generated power domain name follows the `<domainPrefix><delimiter><supplySetName> format`.

In the following example, the power net `VDD` and `VSS` are registered as a supply set, also a top-level power domain is registered as `PD_TOP`, which is associated with this supply set having power nets and ground nets as `VDD` and `VSS`, respectively.

```
;;; Supply Sets
     supplySets (
        (nil
        supplySetName    "SS_VDD_VSS"
        power    "VDD"
        ground    "VSS"
        )
        (nil
        supplySetName    "SS_VDDA_VSSA"
        power    "VDDA"
        ground    "VSS"
        )
        )

    ;;; Power Domains
    powerDomains (
    ( nil
          domainName        "PD_TOP"
          elements          "."
         primarySupplySet    "SS_VDD_VSS"
        includeScope t
        )
    ( nil
          domainName        "PD_LS"
          elements          "I0"
         primarySupplySet    "SS_VDDA_VSS"
        )
    )
```

Note: Specifying supply sets and power domains in the setup is not mandatory if the 1801 import flow is used.

If you do not register power net and ground net pairs, the ground net associated with a power net is automatically traced, consequently, creates a supply set and an associated power domain. Consequently, all the other supply sets are created. This is possible only when there exists a path from a power net to a ground net through a network of transistors and/or two terminal devices. In other cases, you need to register the supply set.

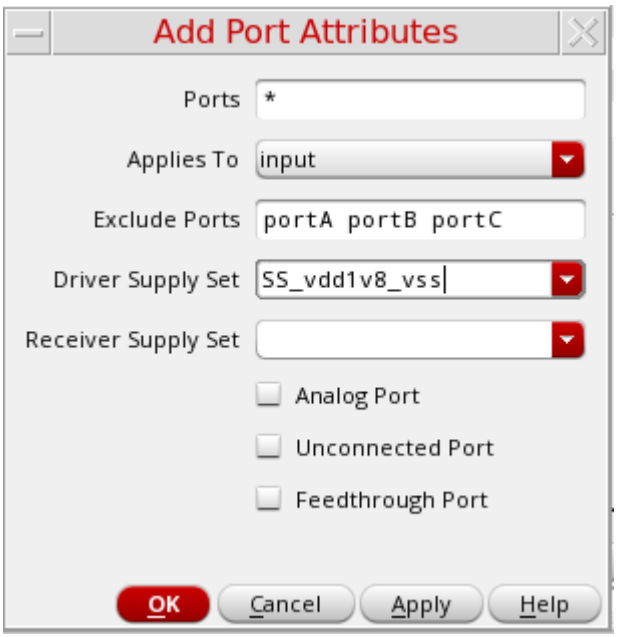
Port Attributes Registration

There are specific design configurations where a boundary port is connected to different blocks that have different supply sets. This leads to ambiguities during the 1801 power intent extraction or the In-Design checks to associate the boundary ports to a power domain.

To register port attributes:

1. Add ports by using the Add Port Attributes form in the *Port Attributes* section on the *Export* tab of the Power Manager Setup form to resolve such an ambiguity. For analog macros, Power Manager offers a mechanism of hierarchical supply traversal till device level to extract power attributes for the boundary ports.
2. Specify the port names or regular expression in the *Ports* field.
3. Define the direction attribute by using *Applies To*.
4. Define the ports for exclusion in the *Exclude Ports* field
5. Select the driver and receiver supply set.

6. Specify if it is the an Analog, Unconnected, and Feedthrough port and click OK.



Alternatively, you can use `portAttributes` in the setup template as shown:

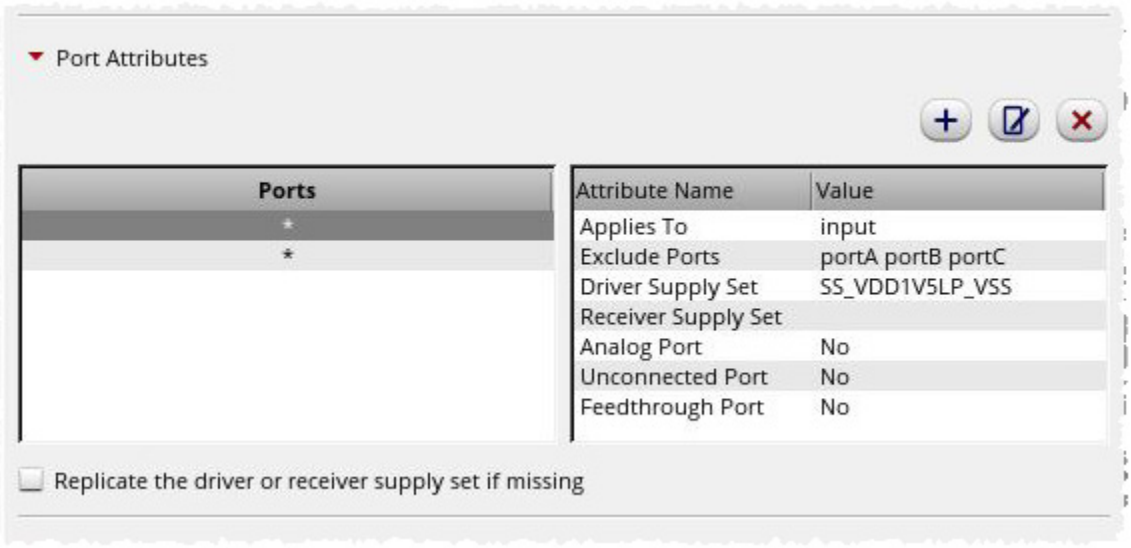
```
portAttributes(
(nil
ports "enable_pin"
driverSupplySet"SS_VDD_GND"
receiverSupplySet"SS_VDD_GND"
isAnalog<t/nil>
isUnconnected<t/nil>))
```

You can specify the following for a particular list of ports defined using `portAttributes`:

- Driver and receiver supply set associated with the digital ports.
- Analog and unconnected ports, where you can add a separate set of ports that do not have the driverSupplySet or receiverSupplySet.

The port attributes specified using `portAttributes` should take the highest precedence and override other definitions. The 1801 power intent extraction or the in-design checks consider the attributes specified for ports by using `portAttributes`, if registered in the setup template. If you specify both the driver and the receiver supply set for a port, they are used during extraction. If you specify only a driver supply set for input port, the 1801 power

intent extraction or the In-Design Checks use it and trace only the receiver supply set and conversely.

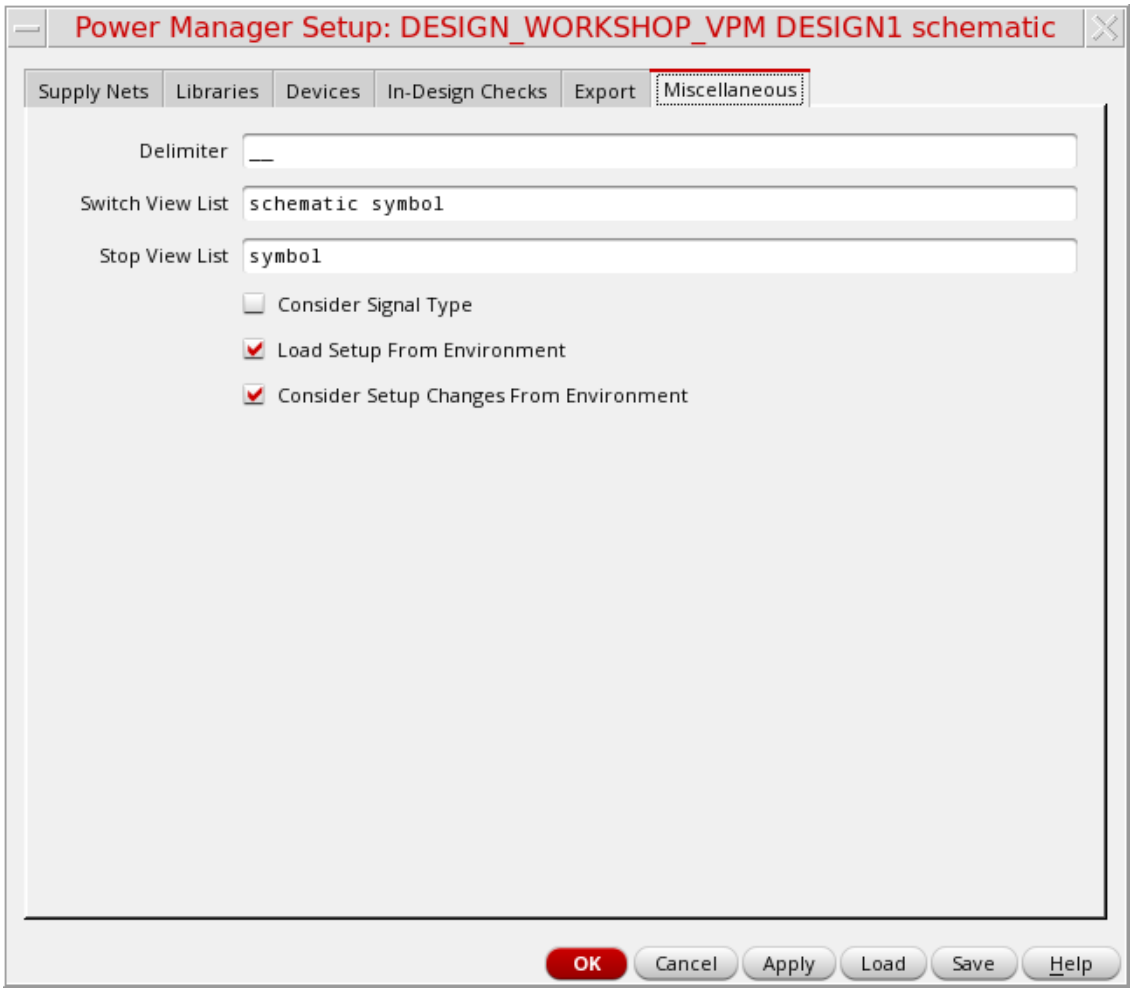


Select the following options in the *Export Options* section to control the extractor behavior for the specific design scenarios:

- *Consider inputOutput terms for internal power criterion*: For details, see allowInoutLDOPins.
- *Consider inputOutput terms for monitor power criterion*: For details, see allowInoutMonitorPins.
- *Consider symmetrical source and drain for supply tracing*: Defines a port with a predefined supply set or power domain that can be checked for the correct or compatible connection at the receiver side. By default, this option is deselected. Supply tracing is done by Power Manager for MOS terminals (Source and Drain) by considering them asymmetric in nature. When this option is selected, Power Manager traces through all devices irrespective of the way they are connected in the design.
- *Use anonymous supply for top level ports*: Enables the use of an anonymous supply for the top-level ports instead of the default supply set.

Miscellaneous Settings

Specify the generic settings on the *Miscellaneous* tab.



- *Delimiter*

  Used for specific notations, for example, redirect netSets created during the 1801 import flow, automatically generated supply set and power domain names, and so on.

- *Switch View List* to `schematic symbol`: It controls the order of the traversal of the design hierarchy during power intent extraction.

- *Stop View List* to `symbol`: It defines the stop point where the design traversal must be stopped by the extractor during power intent extraction.

  `switchViewList` and `stopViewList` can be specified in the Power Manager setup template or defined using the switchViewList and stopViewList environment variables.

- *considerSignalType* to ignore or consider the `sigType` attribute of nets for detecting the supply nets during power intent extraction. By default, this option is set to `nil` and the attribute is not considered during name-based registration for detecting supply nets. However, if set to `t` and there is a conflict between the `sigType` attribute and name-based registration, name-based registration takes precedence.

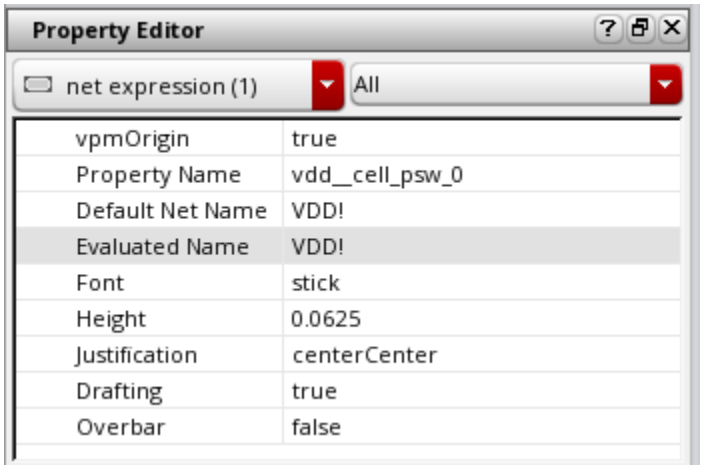
  - Setting the signal type to identify the power nets and ground nets

    Each net has a `Signal Type` attribute that when set to `power` or `ground` can be used to identify the net as a power net or a ground net, respectively. The default signal type of a net is `signal`. During the automatic extraction, Power Manager looks for the `considerSignalType` attribute. When set to `true`, it creates power domains considering all the nets that have the signal type defined as power or ground.

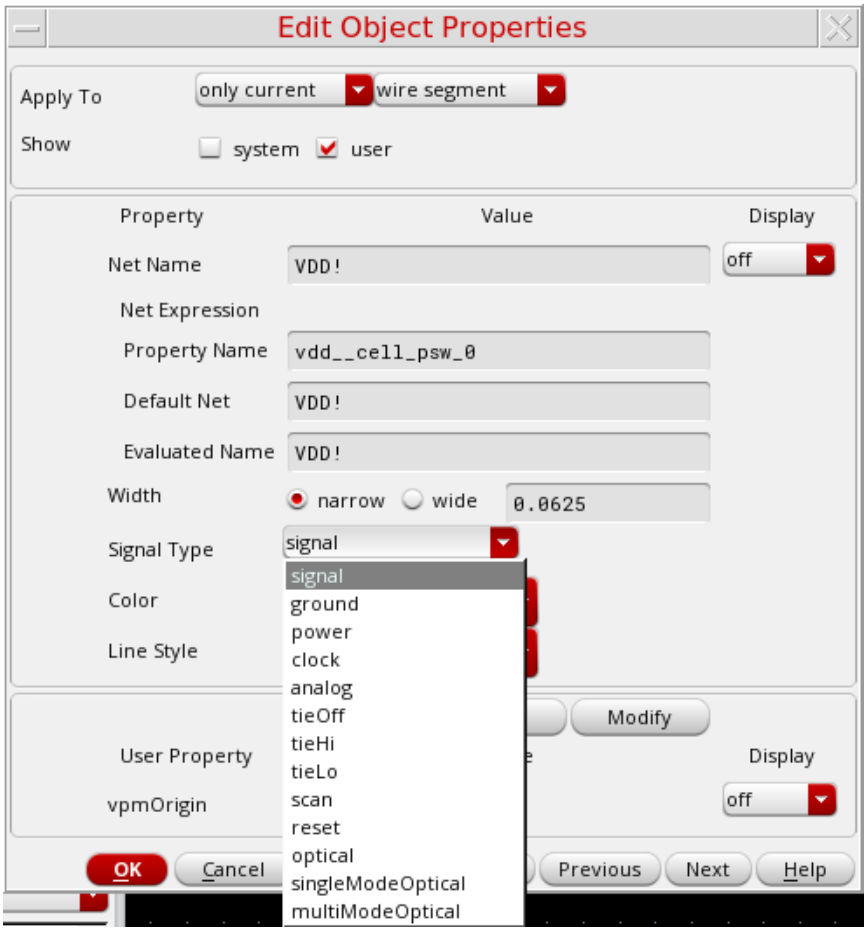
    Note: If there is a conflict between the sigType attribute and name-based registration, name-based registration takes precedence.

    You can set the signal type for pins or nets in your design in one of the following ways:

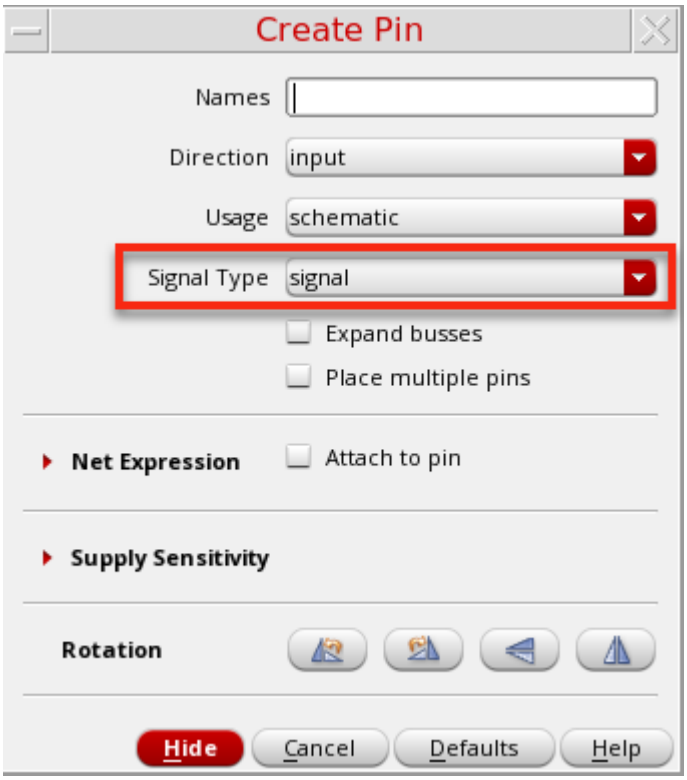
    - Select a pin or net on the schematic and in the Property Editor assistant, update the *Signal Type* property.



- Right-click a net and choose *Properties* to open the **Edit Object Properties** form. Update the *Signal Type* property in this form.



- Specify the appropriate signal type while creating a new pin by using the **Create Pin** form. The *Signal Type* field, which is by default set as `signal`, specifies the pin type.



For the power pins, you can set the value in this field as `power`.

If you use a specific set of names, such as, `vdd`, `vdd!`, or `VDD`, for the power pins in your designs and `vss`, `vss!`, or `VSS`, for the ground pins, it is recommended that you register those names by using the ciRegisterNet API in `.cdsinit` file. If you use a registered name for a pin on the **Create Pin** form and the signal type is set as `signal`, the default value, the tool automatically sets the signal type of that pin as `power` or `ground` respectively. In that case, you need not specify the signal type explicitly for each new pin or net created in the design. Later, if you do not need to use the registered names, you can set the registration as `nil`.

The name-based registration in the setup file template for the Power Manager is as follows:

```
;;; Supply Nets
    supplyNets (nil
        power (nil
            names ("vdd" "avdd" "dvdd" "VDD" "AVDD" "DVDD")
            regExprs ("[vV][dD][dD]" "[vV][cC][cC]")
```

```
            )
            ground (nil
                names ("vss" "avss" "dvss" "VSS" "AVSS" "DVSS")
                regExprs ("[vV][sS][sS]" "[gG][nN][dD]" "gnd*")
```

- Select *Load Setup From Environment* and *Consider Setup Changes From Environment* to load setup or change the environment setup.

Setup File Template

```
lpSetupOptions = '(nil

    ;;; Supply Nets
    supplyNets (nil
        monitor (nil
            names ("VDD_PROBE" "VSS_PROBE")
            regExprs ("VDD_PROBE*" "VSS_PROBE*")
        )
        excludePG (nil
            names ("OUTVDD")
            regExprs ("OUTVDD*")
        )
        ground (nil
            names ("Vss" "Vssa" "Vss_Int")
            regExprs ("[vV][sS][sS]")
        )
        power (nil
            names ("Vdd_1v8" "Vdd_3v3" "Vdd_Int")
            regExprs ("[vV][dD][dD]" "[vV][cC][cC]")
        )
    )

    ;;; Supply Net Voltages
    netVoltages (
        ("Vss_Int" (0.0))
        ("Vdd_1v8" (1.8 1.0))
        ("Vssa" (0.0))
        ("Vdd_3v3" (3.3))
        ("Vss" (0.0))
        ("Vdd_Int" (3.3 1.0))
    )
```

```
;;; External Switchable Nets
externalSwitchableNets nil

;;; Devices
devices (nil
    cap (
        ("analogLib" "cap" nil)
    )
    pfet (
        (nil "pmos*" nil)
        ("analogLib" "pmos*" nil)
        ("gpdk*" "pmos*" nil)
    )
    nfet (
        (nil "nmos*" nil)
        ("analogLib" "nmos*" nil)
        ("gpdk*" "nmos*" nil)
    )
    short (
        ("analogLib" "res" nil)
        ("analogLib" "rcwireload" nil
            (nil
                shortedTerminalMap  (("t1" "t2") ("t3" "t4" "t5"))
            )
        )
    )
    diode (
        ("analogLib" "diode" nil
            (nil
                pTerm               "PLUS"
                nTerm               "MINUS"
            )
        )
    )
)

;;; Transistor Terminal Names
txTermTypeNames (nil
    substrate "S BULK well SUB"
    bulk      "B BULK well SUB"
```

```
    emitter   "e E emitter"
    collector "c C collector"
    source    "s S source SOURCE Source"
    drain     "d D drain DRAIN Drain"
    gate      "g G gate GATE Gate"
    base      "b B base BASE Base"
)

;;; Library Files
libFiles (
    "./DESIGN1/LIBRTY/fast_vdd1v0_basicCells.lib"
    "./DESIGN1/LIBRTY/macro.lib"
    "./DESIGN1/TECHUPF/diode.upf"
    "./DESIGN1/TECHUPF/lshifter.upf"
    "./DESIGN1/TECHUPF/isolation.upf"
    "./DESIGN1/TECHUPF/pswitch.upf"
    "./DESIGN1/LIBRTY/macro.lib"
    "./DESIGN1/LIBRTY/VPMWS_ANA_OpAmp.lib"
)

;;; Text Files Specifying Library Files
libInputFiles nil

;;;  Cell 1801 File Bindings
cell1801Bindings nil

;;;  User Macro Cells
userMacroCells nil

;;;  Reference Library-Cells
refLibCells nil

;;; Supply Sets
supplySets (
    (nil
        supplySetName  "SS_vdd3v3_vss"
        power          "Vdd_3v3"
        ground         "Vss"
    )
    (nil
        supplySetName  "SS_vdd1v8_vss"
```

```
        power           "Vdd_1v8"
        ground          "Vss"
    )
    (nil
        supplySetName   "SS_vddsw_vss"
        power           "Vdd_Int"
        ground          "Vss_Int"
    )
)

;;; Power Domains
powerDomains (
    (nil
        domainName          "PD_vddsw_vss"
        elements            "I4"
        primarySupplySet    "SS_vddsw_vss"
    )
    (nil
        domainName          "PD_vdd3v3_vss"
        elements            "."
        primarySupplySet    "SS_vdd3v3_vss"
        includeScope        t
    )
    (nil
        domainName          "PD_vdd1v8_vss"
        elements            "I0 I2"
        primarySupplySet    "SS_vdd1v8_vss"
    )
)

;;; Port Attributes
portAttributes nil

;;; Maximum number of similar violations to be reported
maxSimilarViolations1

;;; Use supplyStates
useSupplyStatest

;;; Supply States
supplyStates (
```

```
        ("SS1" ("Vss 0.000000" "Vdd_Int 3.300000" "Vdd_3v3 3.300000" "Vssa
0.000000" "Vdd_1v8 1.800000" "Vss_Int 0.000000"))
        ("SS2" ("Vss 0.000000" "Vdd_Int 3.300000" "Vdd_3v3 OFF" "Vssa 0.000000"
"Vdd_1v8 1.800000" "Vss_Int 0.000000"))
    )

    ;;; Input Voltage Tolerance Lower-Bound
    inputVoltageToleranceLowerBound-0.100000

    ;;; Input Voltage Tolerance Upper-Bound
    inputVoltageToleranceUpperBound0.100000

    ;;; Input Ground Voltage Tolerance Lower-Bound
    inputGroundVoltageToleranceLowerBound-0.100000

    ;;; Input Ground Voltage Tolerance Upper-Bound
    inputGroundVoltageToleranceUpperBound0.100000

    ;;; In-Design Checks Filter Patterns
    lprcFilters nil

    ;;; Missing Level-Shifter Check Severity
    missingLSCheckSeverity"error"

    ;;; Incompatible Level-Shifter Check Severity
    incompatibleLSCheckSeverity"error"

    ;;; Redundant Level-Shifter Check Severity
    redundantLSCheckSeverity"warning"

    ;;; Protected Level-Shifter Check Severity
    protectedLSCheckSeverity"info"

    ;;; Unprotected Level-Shifter Check Severity
    unprotectedLSCheckSeverity"error"

    ;;; Unreliable Level-Shifter Check Severity
    unreliableLSCheckSeverity"ignore"

    ;;; Floating Level-Shifter Check Severity
    floatingLSCheckSeverity"error"
```

```
;;; Always Enabled Level-Shifter Check Severity
alwaysEnabledLSCheckSeverity"error"

;;; Unused Enable Level-Shifter Check Severity
unusedEnableLSCheckSeverity"error"

;;; Missing Isolation Check Severity
missingISOCheckSeverity"warning"

;;; Incompatible Isolation Check Severity
incompatibleISOCheckSeverity"warning"

;;; Redundant Isolation Check Severity
redundantISOCheckSeverity"warning"

;;; Incompatible Bulk Check Severity
incompatibleBulkCheckSeverity"error"

;;; Report isolation violations for always-on-switched domain crossings
reportOnOffISOViolationsnil

;;; Check ports without any portAttributes definition in setup
checkPortsWithoutAttributesnil

;;; Report violations for all net voltages
reportLPRCViolationsForAllNetVoltagesnil

;;; Prefixes for Supply Net netSet Props
powerNetPropPrefix  "vdd"
groundNetPropPrefix "vss"
pwellNetPropPrefix  "vdd_sub"
nwellNetPropPrefix  "vss_sub"
deeppwellNetPropPrefix"vdd_sub2"
deepnwellNetPropPrefix"vss_sub2"

;;; Project MLDB Library Name
projectMldbLibName""

;;; Reference MLDB Library Names
referenceMldbLibNames nil
```

```
    ;;; replaceExistingLibs
    replaceExistingLibst

    ;;; Replicates missing driver_supply_set or receiver_supply_set for input,
output ports respectively.
    replicateMissingDriverReceiverSupplySetnil

    ;;; Power Domain Name Prefix
    powerDomainNamePrefix"PD"

    ;;; Enable Automatic Creation of Power Domains
    autoCreatePowerDomainsnil

    ;;; Consider signal type for detecting PG nets
    considerSignalTypenil

    ;;; Set the mode (design/auto) for extraction
    extractionMode"design"

    ;;; Set how the PST/Power State are Created
    powerStateCriteria"Conservative"

    ;;; Set if All Off PST/Power Stated are Created
    includeAllOffStatesnil

    ;;; Control printing of nets identified as supply nets
    printSupplyNetInfonil

    ;;; Set if inout pins are allowed as LDO pins
    allowInoutLDOPinsnil

    ;;; Set if inout pins are allowed as monitor pins
    allowInoutMonitorPinsnil

    ;;; Delimiter for use in power intent objects like Supply Set Names
    delimiter"__"

    ;;; View Name list for hierarchy elaboration
    switchViewList"schematic symbol"
```

```
    ;;; Stack transistor device param names
    stackTransistorDeviceParamNames""

    ;;; View Name list for pruning the hierarchy elaboration
    stopViewList"symbol"

    ;;; Set if the environment options are to be loaded
    loadEnvironmentOptionsnil

    ;;; Set if the options not overridden by the user are to be updated
    pushUnmodifiedEnvOptionst

    ;;; Use anonymous supply set for non-annotated top level ports
    topPortsHaveAnonSupplyt
)
```

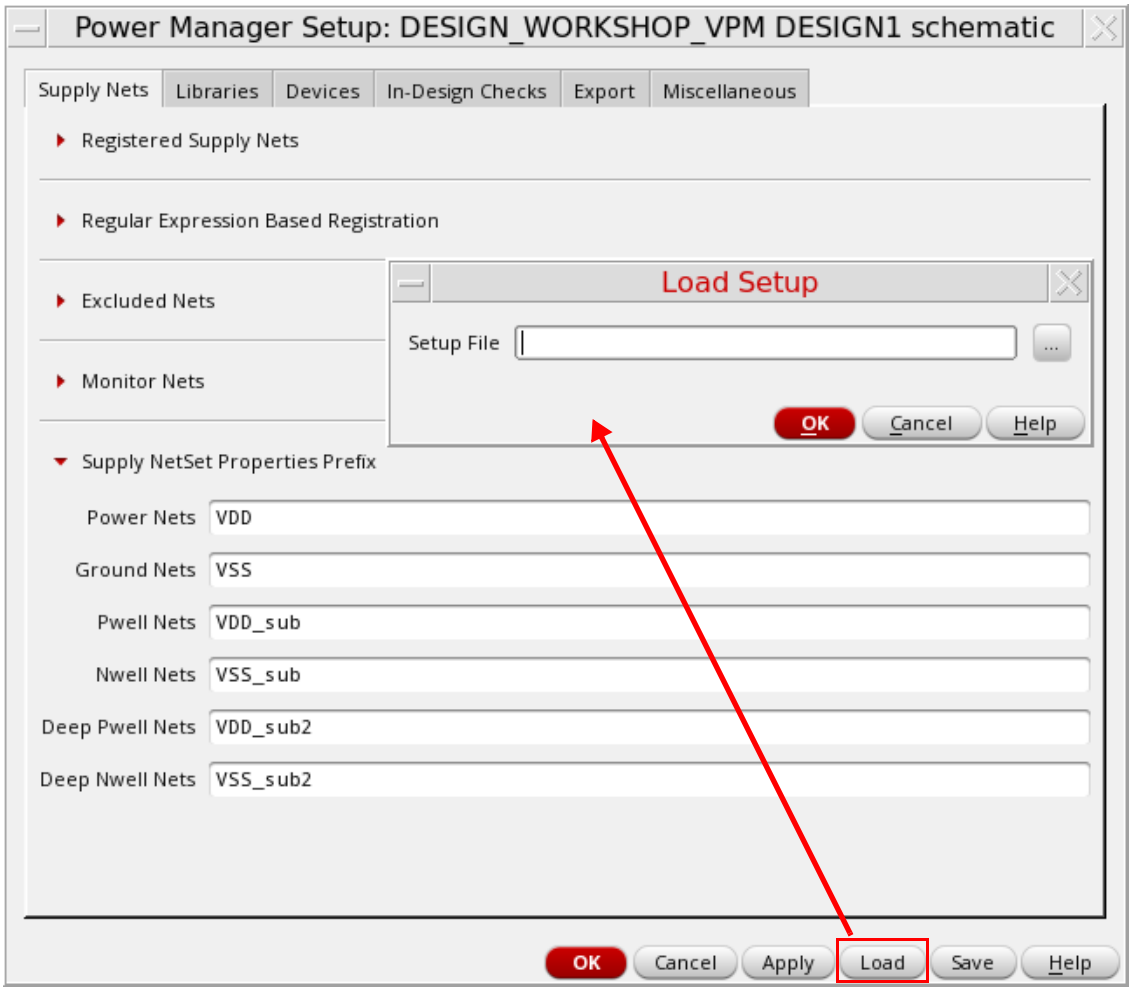
Loading Power Intent Extraction Options from a File

If you have the power intent extraction setup options in a template file, you can load them and reuse them for the same or a different design.

To load the extraction options,

1. In the Power Manager toolbar, click *Power Manager Setup*.

2. In the Power Manager Setup form, click *Load* to select the template file.



3. Browse the file. The tool reads all the extraction options and loads the settings.

   Note: The format of the loaded setup file should match the format of a standard setup file.

4. To apply the changes, click OK.

Important

The Load option adds setup information from the file to the setup form. To put on the changes, click *Apply*.

The setup can also be imported by using the vpmImportPowerIntentSetup SKILL function. This can be used for batch mode processing. The common settings can also be saved in

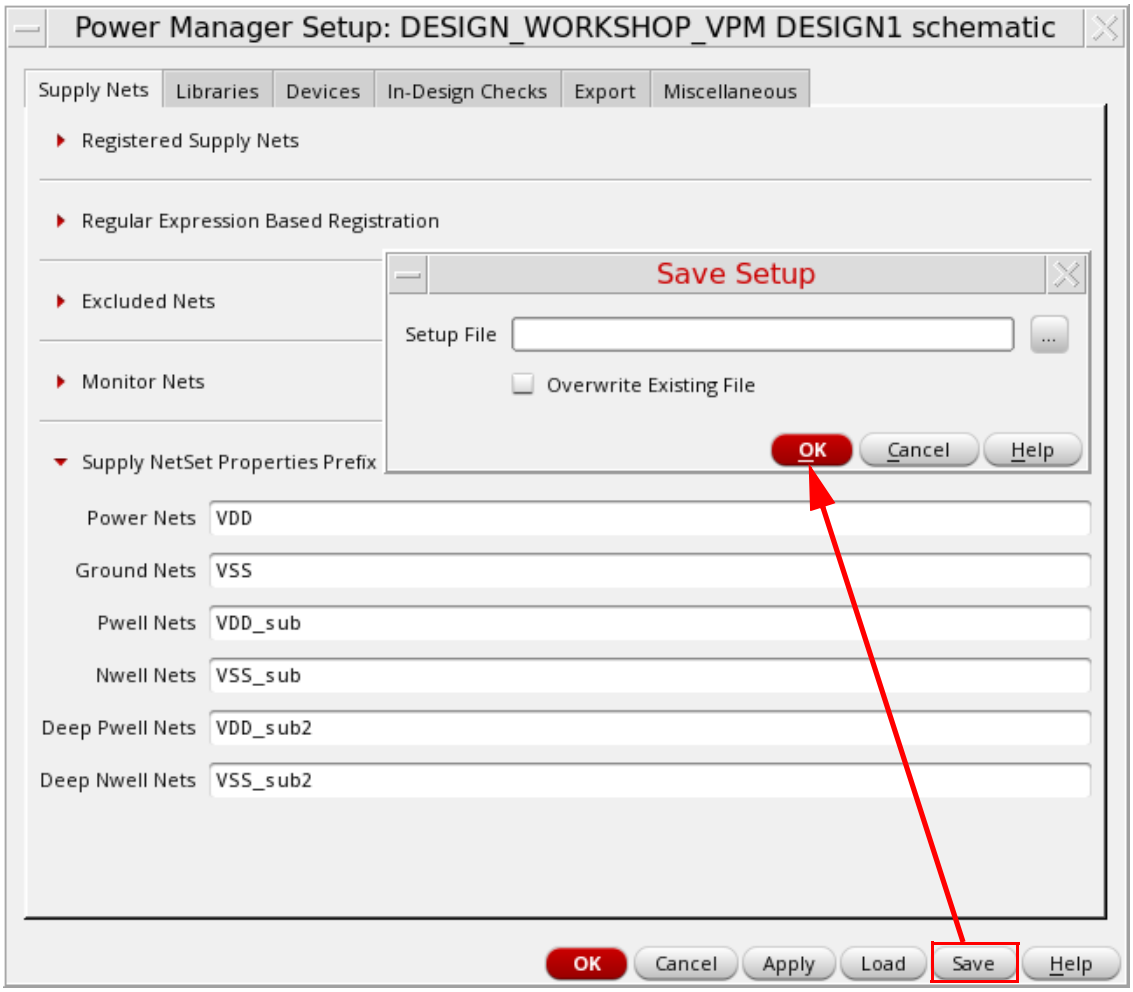
`.cadence/dfII/vpm/lpSetup.il` without the need to reload the setup. These settings can be referenced in any cellview.

Saving Power Intent Extraction Options to a File

You can save the power intent extraction setup options in the setup form to a template file and reuse them for the same or a different design.

To save the extraction options,

1. In the Power Manager toolbar, click *Power Manager Setup*.
2. In the Power Manager Setup form, click *Save*.



3. Browse the file. The tool saves all the extraction options and settings to the file.

4. To apply the changes, click OK.

Important

A confirmation dialog is displayed when you click the Cancel button of the Setup form. It ensures that the unsaved data can be saved in time.

4

Importing the Power Intent of a Design

If the power intent for a design is available in the required (1801) formats, you can import it to update the design with the specified power intent. For this, it is required that the cellview is editable. You can import the power intent to specify it for a cell being instantiated in the design. If you create instances of an IP block in the design for which power intent is available in a 1801 or Liberty file, you can import the file and update the design as per the power intent in the file. This helps in correctly connecting the power domains of the IP to the power domains of the top-level design. In addition, you can register special low power cells that are imported from the library.

The chapter includes the following sections:

Import Flow

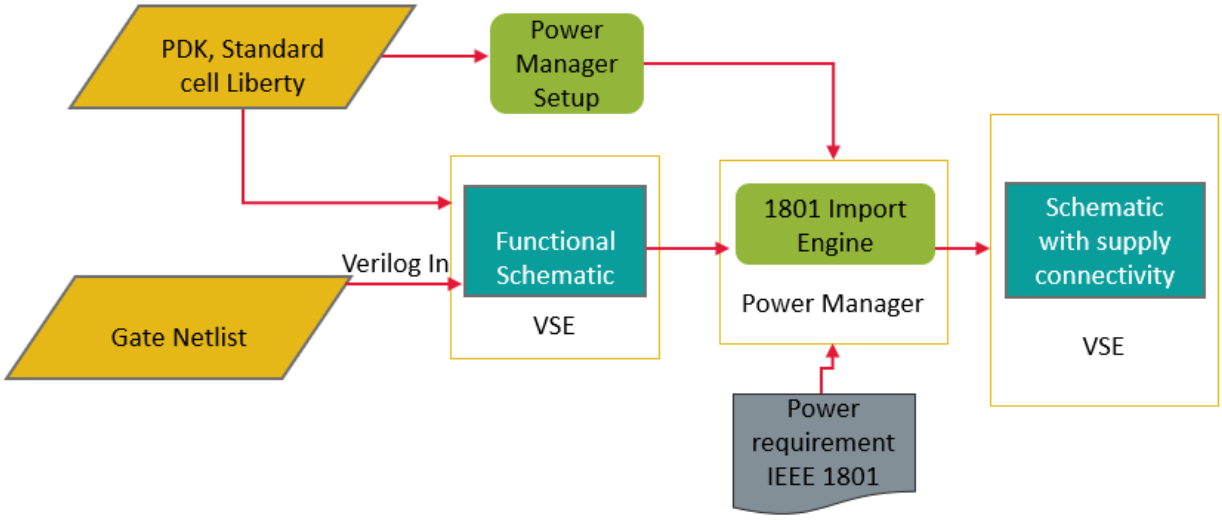
You can import the power intent and utilize it in the following scenarios:

- Use the power intent specified for a design in the top-down power propagation scheme. If you have a hierarchical schematic design for which the power intent is available in a 1801 file, you can import the 1801 file and update the power connectivity of each hierarchical block used in the design as per the power intent. This enables in correctly connecting the power domains of the IP to the power domains of the top design (SOC or a chip).
- If you have a digital block with a Verilog netlist available from a digital place and route tool, supply connectivity can be built for this block by using the Cadence Verilog Import

flow along with the initial imported power intent available for SOC where the digital block is integrated. This defines the power requirements across digital and analog boundaries.

- Register special low power cells that are imported from a 1801 special cell definition file. The registered details are consumed by all the designs that use this file for creating the power connectivity of the cells as per their power domains. The special cell definition file is registered to be included as per the setup. For details, refer to Supply Set and Power Domain Registration.

The illustration below gives an overview of the import flow.

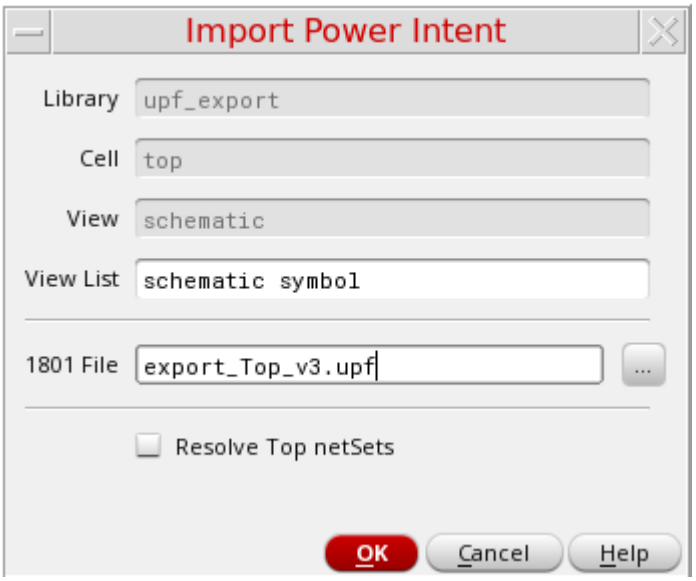


Import Flow in Virtuoso Power Manager

Here are the key stages of the import flow.

- Traversal of design hierarchy to find supply net expressions and explicit terminals.
- Standard and special cells identification using Liberty models or the 1801 special cell definition file.
- Reading the input 1801 power specification.
- Creation and optimization of the redirected netSets.
- Creation of the top-level supply ports and supply nets.

To import the power intent for the currently open cellview, click *Power Manager – Import Power Intent*. It opens the Import Power Intent form. Browse to open the 1801 file.



The currently opened cellview details are specified in the form automatically. The *View List* specifies the switch view list sequence as mentioned in the setup. For details, refer to Miscellaneous Settings. Specify the name of the 1801 file in the *1801 File* field. Alternatively, browse to select the 1801 file to import the power intent for the design.

The *Resolve Top netSets* check box controls the creation of the global net along with the port creation. Select the check box if you do not want the global net to be created along with the associated supply ports defined in the input 1801 file. A local net is created that is associated to each supply port created in the schematic. This mode can be used during the 1801 import for IP integration and extract power intent at the top level.

Deselect the check box if you want a global net expression to be created instead of a local net that is associated with supply ports. Also, corresponding to each unique supply net that is defined in the 1801 file, pins are created. The pin creation is controlled by createExtractionLogFile.

The following illustration shows the output for the supply port VDDA import when `createPinsOnImport` has been set and *Resolve Top netSets* is deselected.

Important

Alternatively, the vpmImportPowerIntent SKILL function has been provided to import the power intent information for a cellview.

The following tasks constitute the import flow in Power Manager:

- The power intent is read from the 1801 file and is updated in the design with all the 1801 commands supported by Power Manager. All unsupported commands or unsupported arguments of the supported commands are flagged in the log.
- Elaboration and traversal of the complete design hierarchy is done to identify the standard and special cells that have an associated Liberty model or a special cell definition file. This is done for the supply and topology recognition. These cells are considered as stop cells and the tool does not traverse through these cells for power/ground nets. Only the top level is read to collect the supply information (inherited nets or explicit terminals), which is further mapped to the power/ground information specified in the Liberty model or the special cell definition file.
- In a hierarchical design for cells that do not have an associated Liberty or a special cell definition, supply nets information is gathered from the different levels of hierarchy. Power Manager traverses down the hierarchy to the level where it finds supply nets of inherited nets (Net Expressions) or explicit supply terminals.
- netSet properties are created in the design hierarchy, wherever required. This enables the top-level block to be instantiated in another top-level SoC schematic. The inherited

pins are created in the block schematic. The inherited terminals have the same name as the supply nets in the 1801 file. For details about the criteria of creating redirected netSet properties, refer to Redirected netset property creation and optimization.

- The tie connections are resolved in the design hierarchy. For details about the resolution of tie connections, refer to Resolving Tie Connections in the Design.
- Supply pins are created corresponding to the power domain nets and global nets during 1801 import by reading the `create_supply_port` and `create_supply_net` commands for a successful LVS check.

  Refer to the following scenarios that are considered while creating the pins.

  - If a pin already exists and it belongs to the same net as mentioned in the 1801 file, it is used.
  - To create an inherited pin, a pin that exists in the same net as mentioned in the 1801 file is used. However, the pin net should be global and should match the power domain net for which the pin needs to be created. The pin is converted into an inherited pin by associating a terminal net-expression with the existing pin. If the existing pin is already an inherited pin but with the different net-expression, the net-expression is replaced.
  - All pins created during the 1801 import are removed when you click *Remove Power Intent*. If any pin existed prior to import and was converted to an inherited pin or the net-expression was changed, it is converted back to a normal pin or the original net-expression is restored, respectively.
  - For the 1801 import flow, pins are not created in the design schematic for the supply nets that do not have a corresponding `create_supply_port` command in the input 1801 file. These supply nets are internally generated supply nets that have a corresponding `create_supply_net` command in the input 1801 file.

  The direction of the pins created aligns with the direction specified in the `create_supply_port` command in the input 1801 file.
- The special cell rules are read as specified in the input 1801 file to create the corresponding supply connections and resolve them for the top supplies. For more information refer to Special Cell Handling for Import flow.

Tip

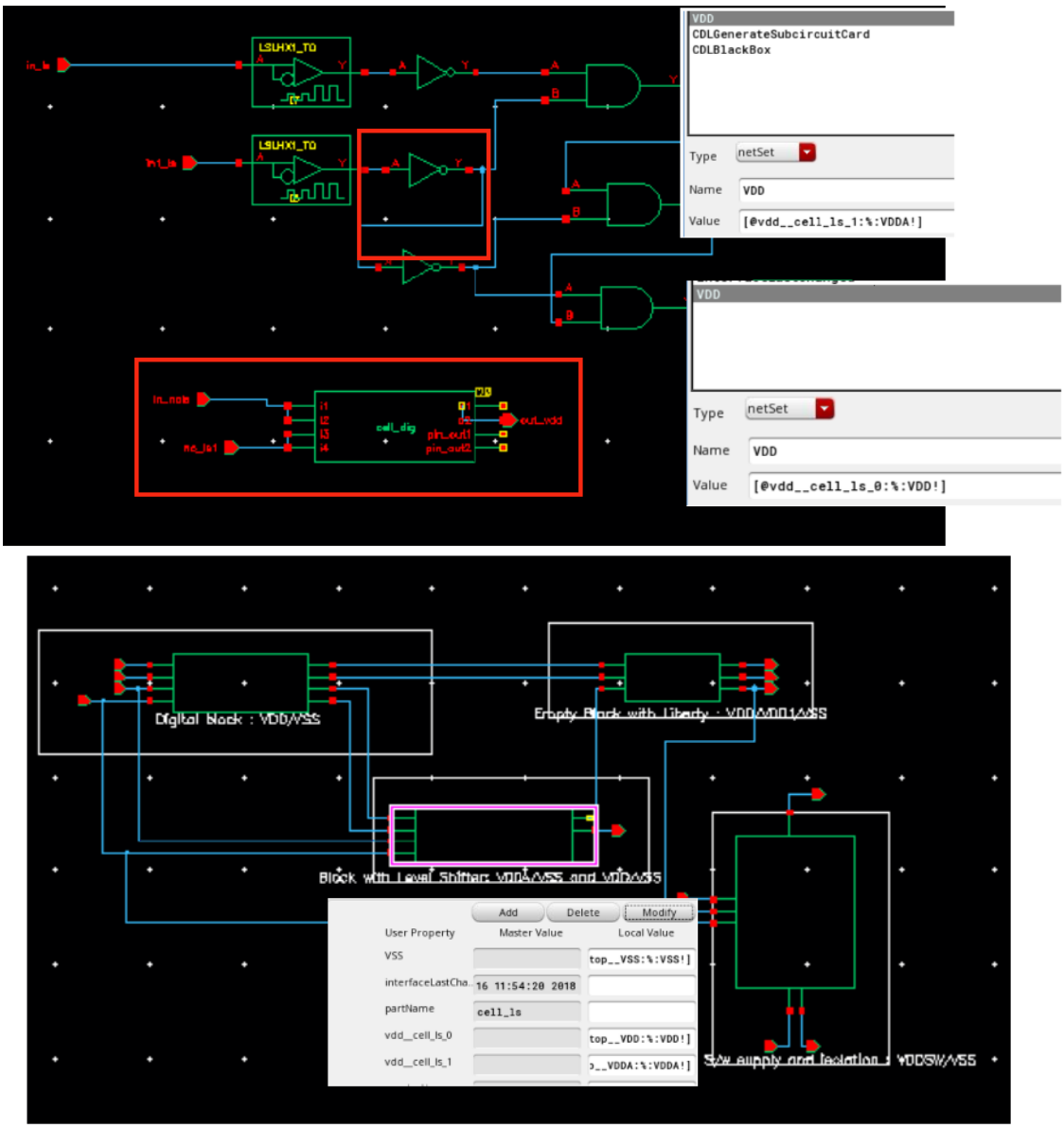
It is recommended that after importing a 1801 file, you run *Check – Hierarchy* or the *File – Check and Save*. These commands create nets corresponding to the netSet properties and set the signal type of power nets and ground nets according to the power intent.

Redirected netSet Property Creation and Optimization

While creating the supply connectivity across the design hierarchy, there could be conflicts where the same net expression property gets resolved to different supply nets at the top. For correct supply connectivity resolution in such scenarios, a redirecting netSet property is created and an optimization algorithm is executed to ensure that the redirection of netSet property happens across the design hierarchy only where required. Redirection is done where having a pure global net expression can cause issues in the supply connectivity across the design hierarchy. This ensures minimum number of netSet properties at the top instance.

Wherever the property name is created, the prefix is added as mentioned in the setup to identify various `pg_type`. Redirected property name is realized as

`[@Prefix_cellname_count:%:topsuplyname!]`. The illustration below represents the creation of redirected netSet property in the design.

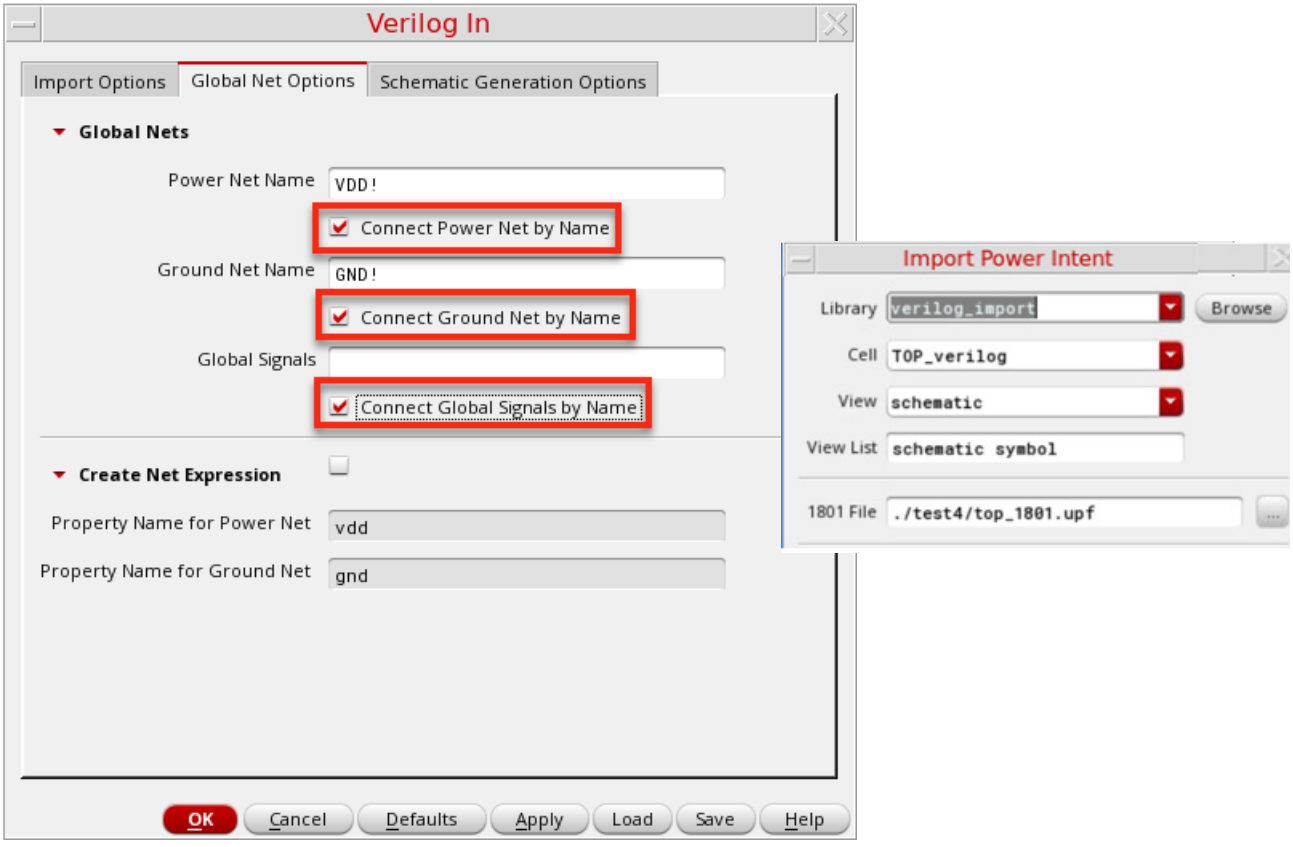


In a design if there is a hierarchy and no Liberty stop cell, Power Manager traverses to the level where net expressions exist. Or, if the net expressions are already redirected to have different property names, it is the reference for the tool to make the connectivity.

Resolving Tie Connections in the Design

In a Verilog netlist if an input pin of a standard cell instance or a special low power cell, such as level shifter or isolation cell instance, is connected to a numerical constant `1'b1` or `1'b0`, the input pin gets connected to a global supply net in the schematic created after Verilog import using Verilog In. The name of the global supply net is specified at the time of Verilog import. After Verilog import, all such input pins connect through a common wire creating tie connections. The label of the common wire matches the global supply net name.

To avoid shorting of nets after importing the 1801 file, it is important that all such pins are connected to wire stubs with a label. However, the wire stubs are not physically connected to each other. To achieve this, use the two connect by name options, *Connect Power Net By Name* and *Connect Ground Net By Name*, during Verilog import. This connects each tied-off input pin with the wire stub that has a label. It enables you to modify the tied-off input pin connections without creating incorrect connectivity in the design through shorting of the nets during the 1801 import.



To resolve the tie connections of various instances, all the labels that require an update are identified during import. The labels on the wire stubs that are attached to the tied-off input

pins are updated. These labels present at the top level or in the lower level block are identified for updates based on the following conditions:

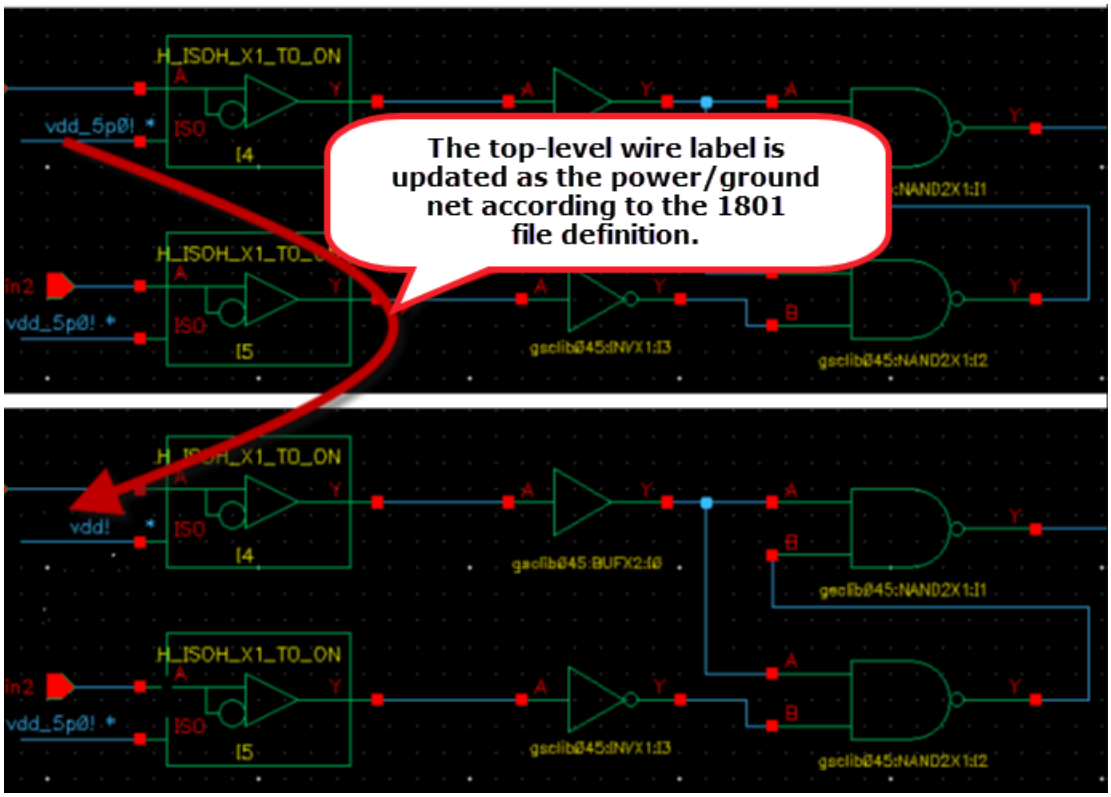
- The label is attached to a wire, which has one end point floating (not connected to anything) and the other end point connected to an instance.
- The wire net is `power`, `ground`, `tieHi`, or `tieLo` according to sigType or name-based registration.
- Wire net has the netType property.

Wire net having the netType property is one of the conditions because it ensures that the label and wire have been added using Verilog In.

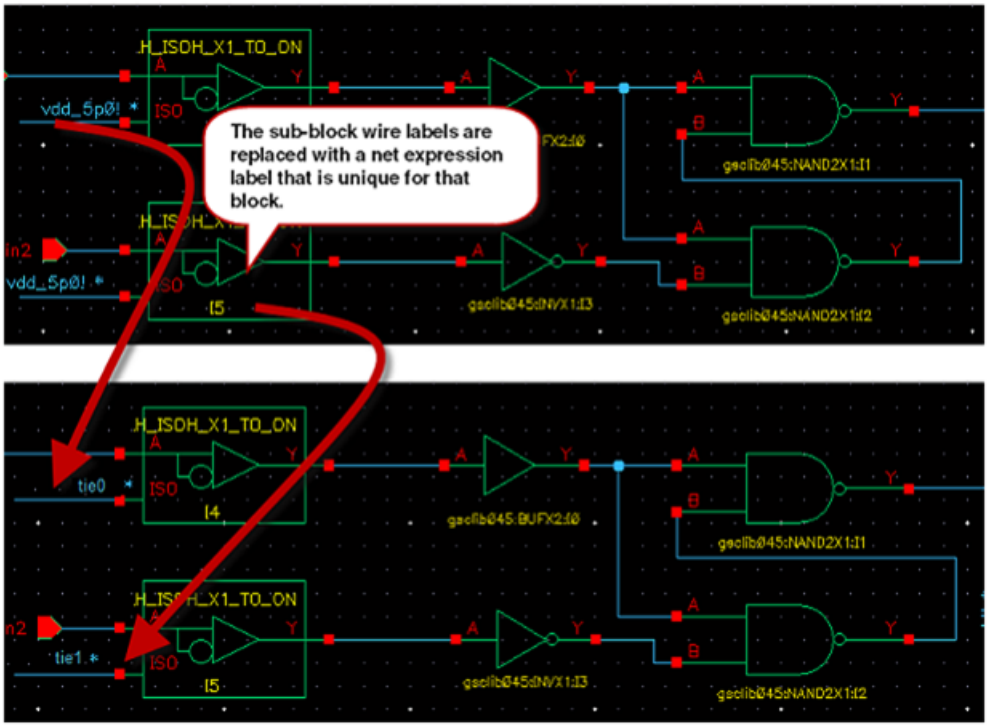
It avoids modification of labels in standard cells, for example, an inverter schematic can have terminal MOS devices with bulk/source/drain terminals connected to wire stubs that have a label, and the wires are of type `power`, `ground`, `tieHi`, or `tieLo`.

Once the target wire label is identified, it is updated based on whether it is a top-level label or a sub-block label.

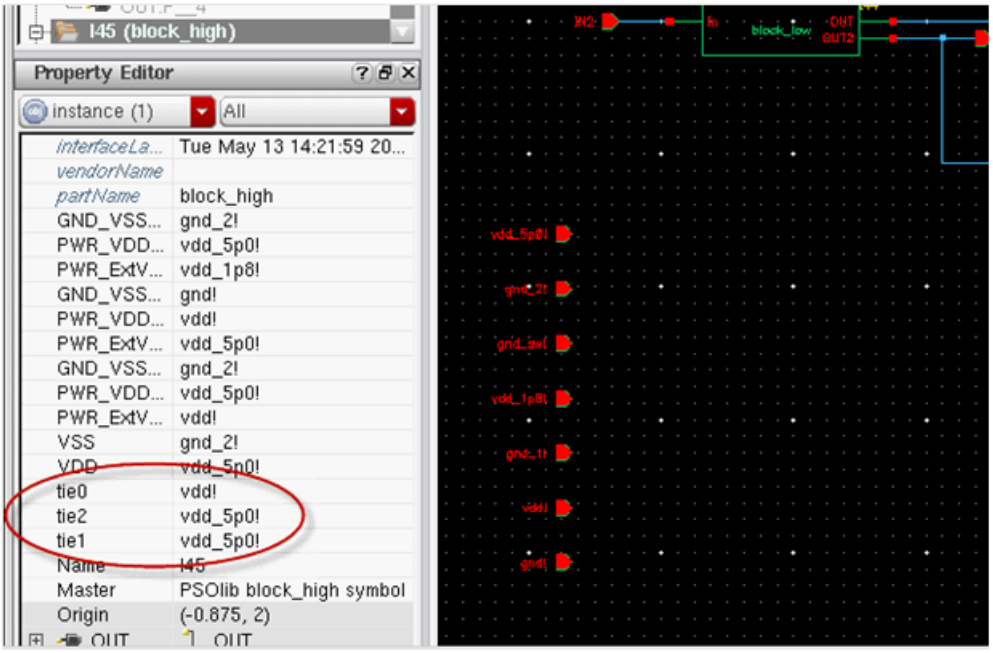
- The top-level wire labels are updated as the power/ground nets according to the 1801 file definition, as shown below.



- The sub-block wire labels are replaced with a net expression label that is unique for that block as shown below.



This net expression can be resolved to its final power/ground value with the help of a netSet property added to the parent instances in the hierarchy above.



The final resolution of the tie connections in schematic designs to the appropriate power and ground nets mentioned in the 1801 file is aligned to the following rules:

- For standard cell instances, the tie connection at the input pin is resolved with the power/ground net of the power domain of the instance.
- For isolation instances, the enable pin with tie connection at the input pin is resolved with the power/ground net of the always-on power domain of the isolation instance.
- For enabled level shifter instances, the enable pin with tie connection is resolved with the power/ground net of the output power domain of the enabled level-shifter instance.
- For isolation and level shifter combo cell instances, the enable pin with tie connection is resolved with the power/ground net of the input power domain of the enabled level shifter instance.

Handling of Low Power Special Cells

The 1801 import flow supports the following low power special cells:

- **Level Shifter**

Import 1801 flow supports different level shifter configurations, such as single rail, dual rail and enabled single /dual rail (combo cell). For each level shifter instance, the load and drivers along with their power domains are determined and the netSet properties are created based on the criteria mentioned below.

- If a level shifter cell is connected to a standard cell or port at both ends, the load and driver domains are used to resolve PG connections.
- If a level shifter instance is a multi-rail cell, the driver domain resolves the input supply connections and the load domain resolves the output supply connections. Level shifter rule specified in the input 1801 file enables in determining the load and the driver domains. The rule is considered to find the supply set based on `input_supply_set` and `output_supply_set` argument. Level shifter rules are specified in input 1801 as:

  ```
  set_level_shifter ls_rule -domain pd_ls -applies_to inputs -rule
  low_to_high -location self -input_supply_set SS_VDD_VSS -output_supply_set
  SS_VDDA_VSS
  ```

  If not specified, Power Manager finds the driver and load supply set for the level shifter by traversing the design path and resolving the connectivity accordingly.
- If a level shifter instance is a single-rail cell, the load domain resolves the output PG connections. `input_signal_level` is required in the Liberty model of the level shifter to identify the cell type as single rail level shifter.
- If a level shifter instance is connected to an isolation cell at one end, the load or driver domain on the other end is determined. If the driver domain is known, it resolves the input supply connections. If the load domain is known, it resolves the output supply connections.

  Note: A single-rail level shifter cell has only output supply.
- If a level shifter instance is connected to an isolation cell at one end, the power domain for the end connected to the isolation instance is determined from the isolation instance. Therefore, the power domain at the isolation cell intersection is the same as the power domain of the isolation cell itself.

■ Isolation cell

Initially, the load and driver of the isolation cell are traced and then, the netSet properties are created based on the criteria mentioned below.

- If an isolation cell is connected to a standard cell or a port on both the ends, the power domains of the load as well as the driver are used to resolve the PG connections and create netSet properties.

- If an isolation cell instance is a multi-rail cell and has a switchable power or ground pin, then the switchable domain is used to resolve the switchable PG connections. The non-switchable domain resolves the non-switchable PG connections.
- If an isolation cell instance is a single-rail isolation cell, only one of the load or driver domain that is non-switchable is used to resolve the PG connections. The isolation rule specified in input 1801 file assists in determining the load and the driver domains. The rule will be considered to find a supply set based on isolation_supply_set defined in the isolation rule. Isolation rules are specified in input 1801 file as:

  ```
  set_isolation ISO -domain PD_VDD_VSS -location self -elements {I1/In} -
  isolation_supply_set SS_VDD_VSS -isolation_signal I0/en -isolation_sense
  high -location self
  ```

  Note: `isolation_supply_set` is required if `default_isolation` is not mentioned in the power domain stated in the input 1801 file.

Note: You can provide an expression containing multiple pins for enable in the `tech.upf` file for isolation cells and level shifter cells. For example, `define_isolation_cell -cells ISOHX1_OFF -enable "EN1&EN2&!EN3"  -power VDD -ground VSS`.

- **Power Switch**

  For power switch, the load and drivers supply along with the control net are determined and the netSet properties are created accordingly. The power switch rule specified in input 1801 file enables in determining the load and the driver domains. The rule is considered to find a supply set based on `input_supply_port`, `output_supply_port` and `control_port`.

  Power switch rules are specified in the input 1801 file as:

  ```
  create_power_switch I3_I0 -output_supply_port { VDD VDDSW } -input_supply_port
  { ExtVDD VDD } -control_port { PSO    psw_en } -on_state { on ExtVDD {!PSO} }
  -off_state { off PSO } -domain PD_TOP
  ```

  If the `create_power_switch` has a `-instance` option, the name of the switch instance(s) are provided that are covered by the power switch rule. If the `create_power_switch` has a `-domain` option, the power switch rule targets all power switches that are inserted within the domain boundary.

Support of Hierarchical 1801 for Import Flow

The import 1801 flow in Power Manager also supports a hierarchical 1801 input file. A hierarchical 1801 has the `load_upf` and `set_scope` commands which set the scope to each of the specified list of instances and executes the set of 1801 commands in the child 1801 file, specified as the argument of the `load_upf` command. Upon return, the current scope is restored to what it was prior to invocation. Supply nets and ports will be created for

load_upf/set_scope block, based on the `create_supply_net` and `create_supply_port` commands mentioned in the parent 1801 file, along with that, it will also obey the settings for createExtractionLogFile and ResolveTopNets.

For all the instances in the design, domain assignment is based on `create_power_domain` and if no element has the power domain specified, it will belong to default power domain. In case there is no default power domain in the scope, then its upper scope default power domain will be looked at to assign the power domain.

Honoring Command Sequence and Precedence

The traversal of the input 1801 file in the import flow follows a set of the below-mentioned rules related to the command sequence and their precedence.

- Any explicit assignment of the elements (instances) in the `create_power_domain` command takes precedence over the extent that includes elements (instances) as a part of that power domain.

  If there is an existing explicit domain assignment with `create_power_domain` in the elements section, irrespective of the ordering of `create_power_domain`, `set_scope`, or `load_upf` commands, the precedence of scope is defined by the nearest ancestor that has an explicit domain assignment using `create_power_domain`.

  This can be shown in the following case considering the sequence of commands in the input 1801file:

  ```
  set_scope "/"
  create_power_domain PD2 -elements {I0/I2} ….
  create_power_domain PD1 -element { . } ....
  set_scope "I0"
  create_power_domain PD3 -elements {.} ….
  ```

  Result: I0/I2 gets assigned to the power domain PD2 and not PD3

- Explicit assignment of multiple domains to the same instance is an error and 1801 stops the import process. This can be shown in the following case:

  ```
  set_scope "/"
  create_power_domain PD1 -elements {I0} ….
  set_scope "I0"
  create_power_domain PD2 -element { . } ….
  ```

  Result: Error case

- `connect_supply_net` always takes precedence over `create_power_domain`. If there are more than one `connect_supply_net` commands for the same supply net,

the last command in the order is honored. In case of multiple `connect_supply_net` commands, the last command is applicable.

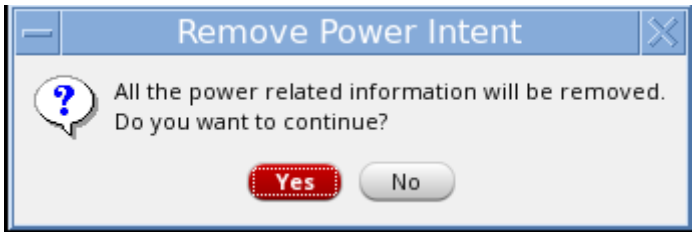
```
set_scope "/"
create_power_domain PD1 -elements {I0} ….
set_scope "I0"
create_power_domain PD2 -element { . }….
```

Result: I0/vdd is connected to VDD2

Removing the Imported Power Intent

To remove the updates done in the design using Import 1801 flow, which include newly created pins, nets, net expressions, and netSet properties on the blocks, you can use the Remove Power Intent option. This removes these objects and restores the previous state of the design before the importing the power intent. This can be accessed from the Power Manager menu and the Power Manager toolbar.

Remove the power intent information from the design by using *Launch – Power Manager – Remove Power Intent.*



Alternatively, you use the vpmRemoveImportedPowerIntent SKILL function for removing the power intent.

5

Running In-Design Checks

The In-Design Checks aim to provide hints to designers for multiple design guidelines or checks and helps in improving the efficiency of the design development cycle. Such checks help checking complex mixed-signal designs that have multiple voltage islands, an increased fusion of analog and digital blocks, more interfaces to check within analog blocks, an increased use of complex power distribution, different modes for achieving power savings, and the use of third-party IPs in various forms. You can also generate signal information for the design, which can be used as a good debugging aid to analyze different power domain crossings in a design.

In-Design checks can be run on any physical design schematic having complete power connectivity. Designs that have the power connectivity introduced post the 1801 Import flow are also suitable candidates for running In-Design checks.

The chapter includes the following sections:

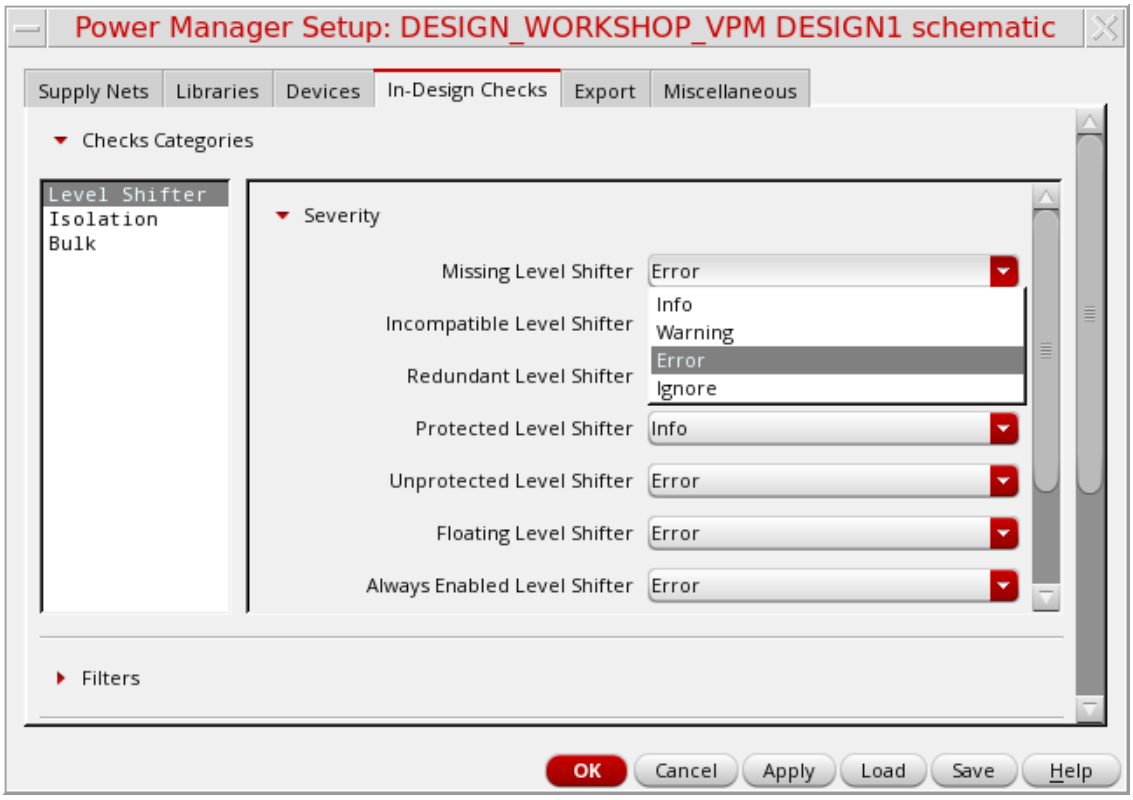
- Defining the Severity of Design Checks
- Running Checks
- Generating Signal Information

Defining the Severity of Design Checks

Power Manager through its In-Design checking functionality performs various static low power structural checks for level shifters and isolation cells. It also aids in checking for invalid connectivity of bulk node for transistors used in the design by performing the bulk checks.

To define the severity for the checks:

1. On the *In-Design Checks* tab of the Power Manager Setup form, select the Level Shifter or Isolation tab.
2. Specify the severity level for various low power checks in the *Severity* section.



The In-Design Checks functionality includes the following structural checks:

- Low power checks
  - Level Shifter Checks
  - Isolation Checks
- Bulk Checks

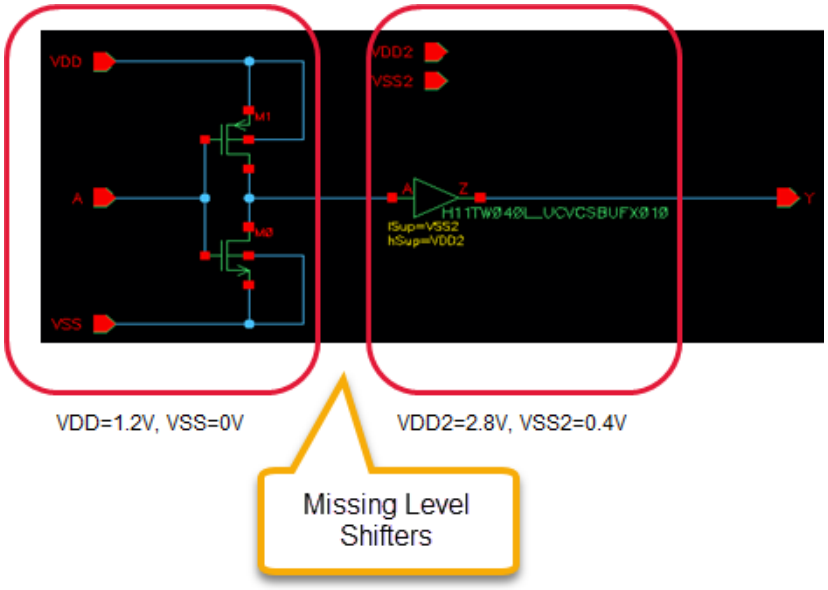
Level Shifter Checks

The following checks are performed for various types of level shifters in a design.

- **Missing level shifters check**

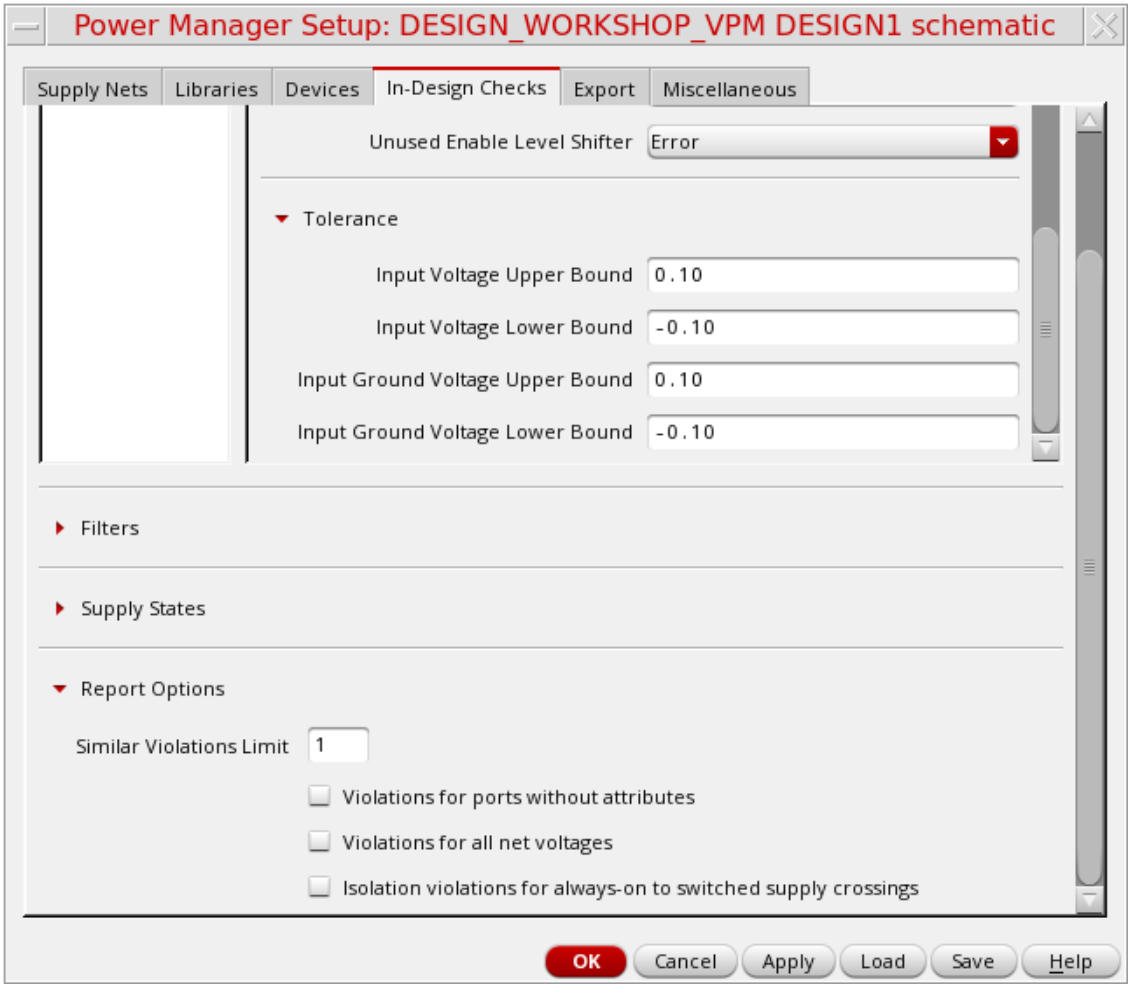
  Checks all the data connections for voltage compatibility. If the voltage values for driver and receiver are not compatible, an error is generated. In this check, the type of missing level shifter (high-to-low or low-to-high) is reported. The driver supplies might be MOS drain terminals, standard cell output, or output macro domain ports. Loads can be MOS gate terminals, standard cell input, or macro domain ports. In addition, domain crossings operating at different voltages without level shifters are reported in one of the following scenarios:

  - The driver power voltage is less than the receiver power voltage by the lower bound input voltage tolerance.
  - The driver power voltage is more than the receiver power voltage by the upper bound input voltage tolerance.
  - The driver ground voltage is less than the receiver ground voltage by the lower bound ground input voltage tolerance.
  - The driver ground voltage is more than the receiver ground voltage by the upper bound ground input voltage tolerance.



  - Tolerance Settings

You can define the lower and upper tolerance values for input and input ground voltages in the *Tolerance* section.



The default value of the lower limit of the input power and input ground voltage is `-0.10`. It should be less than or equal to `0`.

The default value of the upper limit of the input power voltage and input ground voltage is `0.10`. It should be greater than or equal to `0`.

The missing level shifter check also supports the following:

- User-defined Port Attributes Registration for checking the boundary ports.
- Supply States for explicit voltage values.

Note: Power states that are OFF are not considered for reporting missing level shifter errors.

■ Incompatible level shifters check

In this check, all domain crossings at different operating voltages are checked for the level shifters voltage ranges. An error is reported if operating voltages are found incompatible, for example, high level shifters in a low-to-high crossing or conversely.

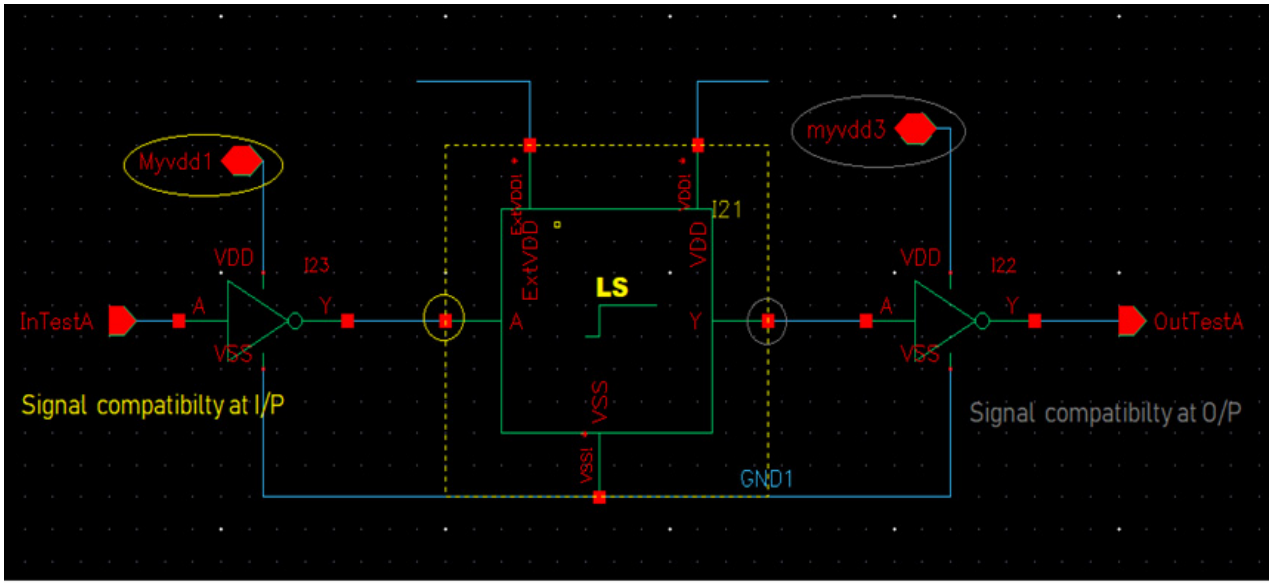
Incompatibility in operating voltages can arise due to the following:

- ❑ Voltage levels of signals

  This checks for incompatible driver or receiver for a data pin with respect to input or output supply voltage range for a level shifter.

- ❑ Voltage levels of supply

  This checks for incompatibility of level shifter supplies with respect to the input or output voltage range supported by the level shifter.

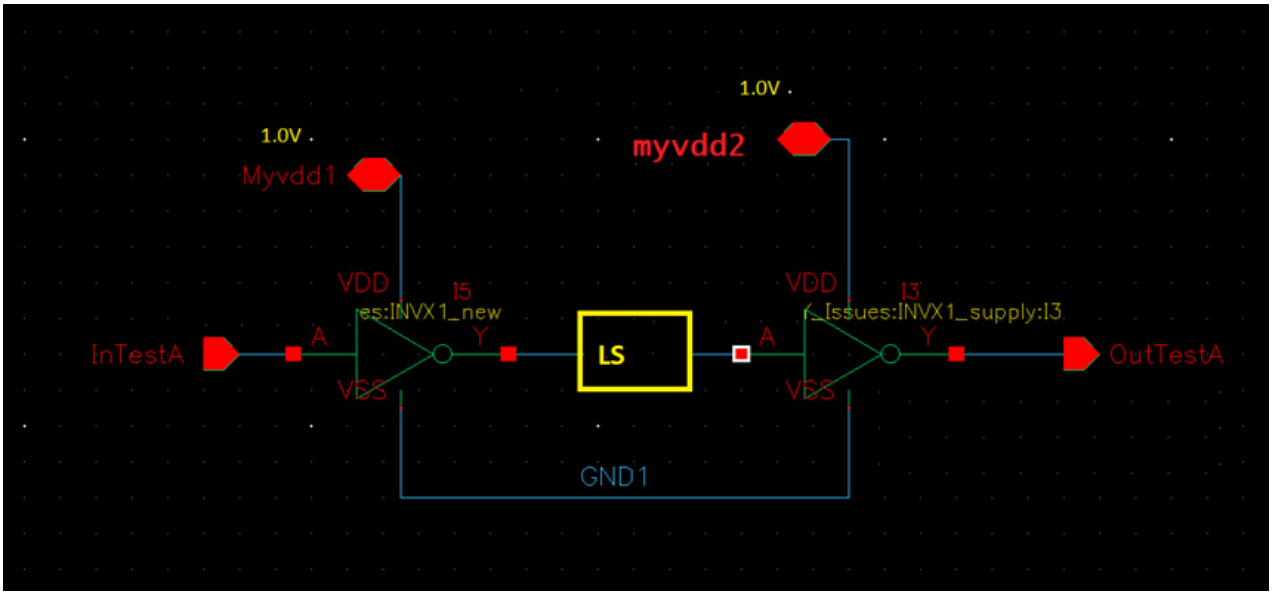


■ Redundant level shifters check

All domain crossings with level shifters at the same operating voltages are reported. In addition, the domain crossings at the macro boundary with a redundant level shifter are reported.

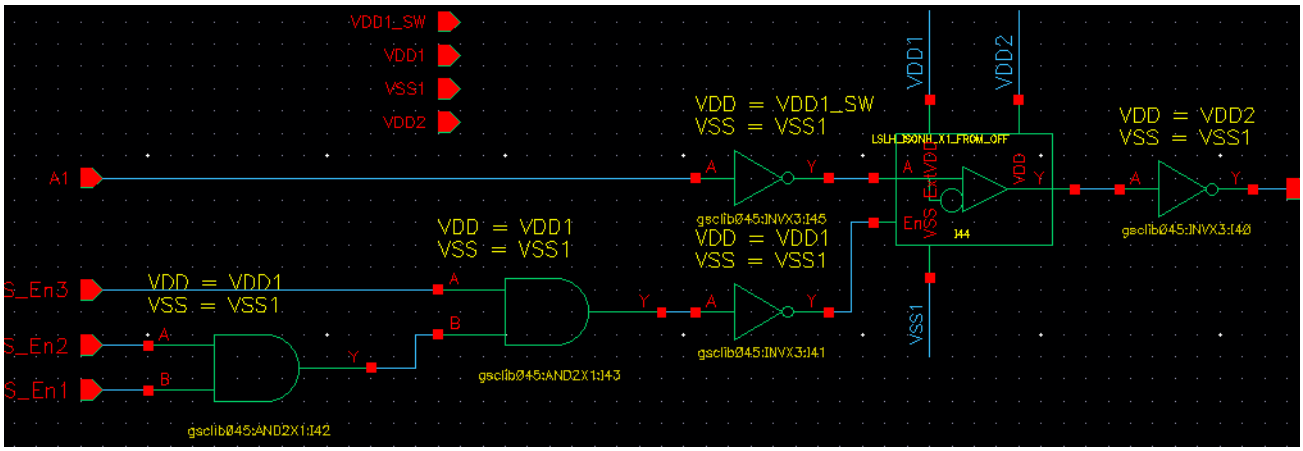
Note: A valid level shifter in one supply state is not considered redundant in any other

power state.



■ Protected level shifters check

The data is protected by using an enable signal. The protected level shifters in design along with their enable condition are highlighted.

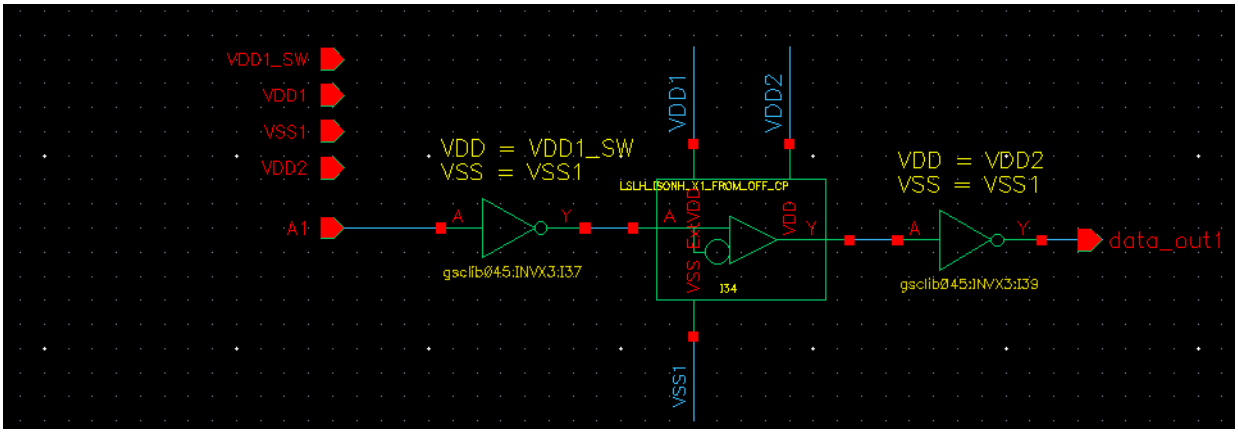


■ Unprotected level shifters check

An unprotected dual rail level shifter is reported when all the following conditions are true:

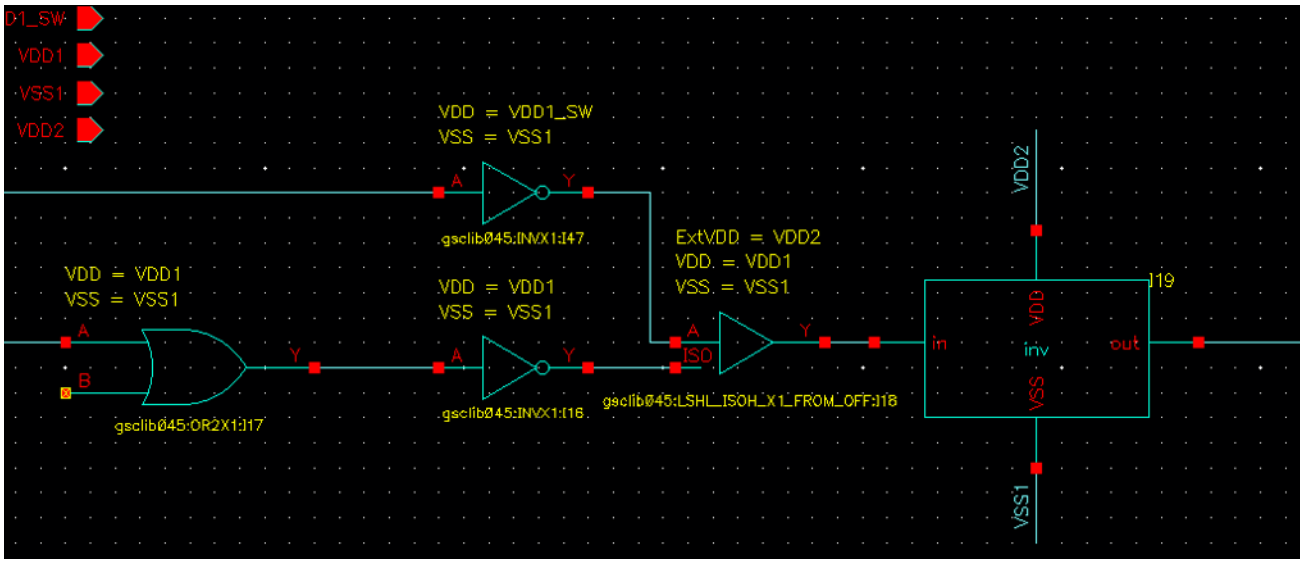
- ❑ Level shifter instance has no enable pin.
- ❑ One domain of the level shifter instance is in the OFF state.

- The second domain of level shifter instance is in ON state.



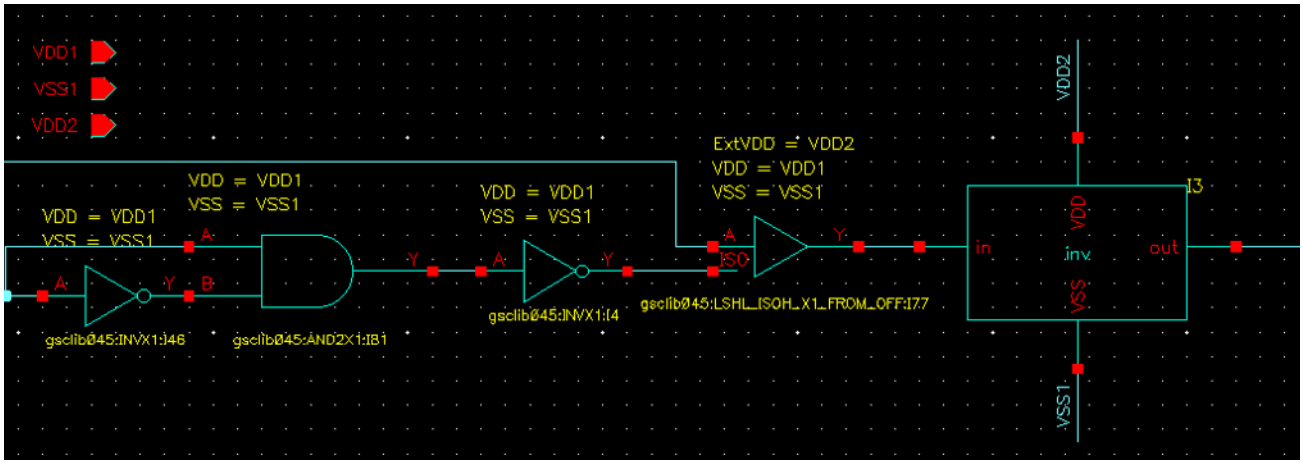
■ Floating level shifters check

The floating level shifters have their enable signal evaluated as floating. During the check, the level shifter is reported if its enable input can be traced to a floating value.



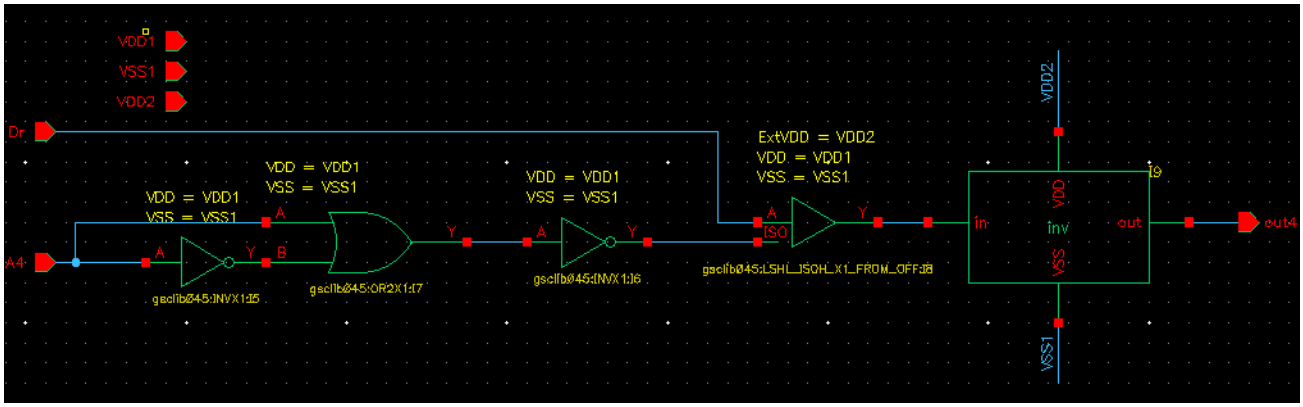
■ Always enabled level shifters check

This check highlights the level shifters that have the enable pin tied to a fixed supply voltage and therefore, the state of the level shifter never changes. It keeps the data output always clamped.



- **Unused enable level shifters check**

  This check highlights a level shifter that has the enable pin tied to a fixed supply voltage of opposite polarity. In this case, the enable pin is not used effectively.The unused enable level shifters have an enable signal tied to a fixed voltage and data output is never clamped.

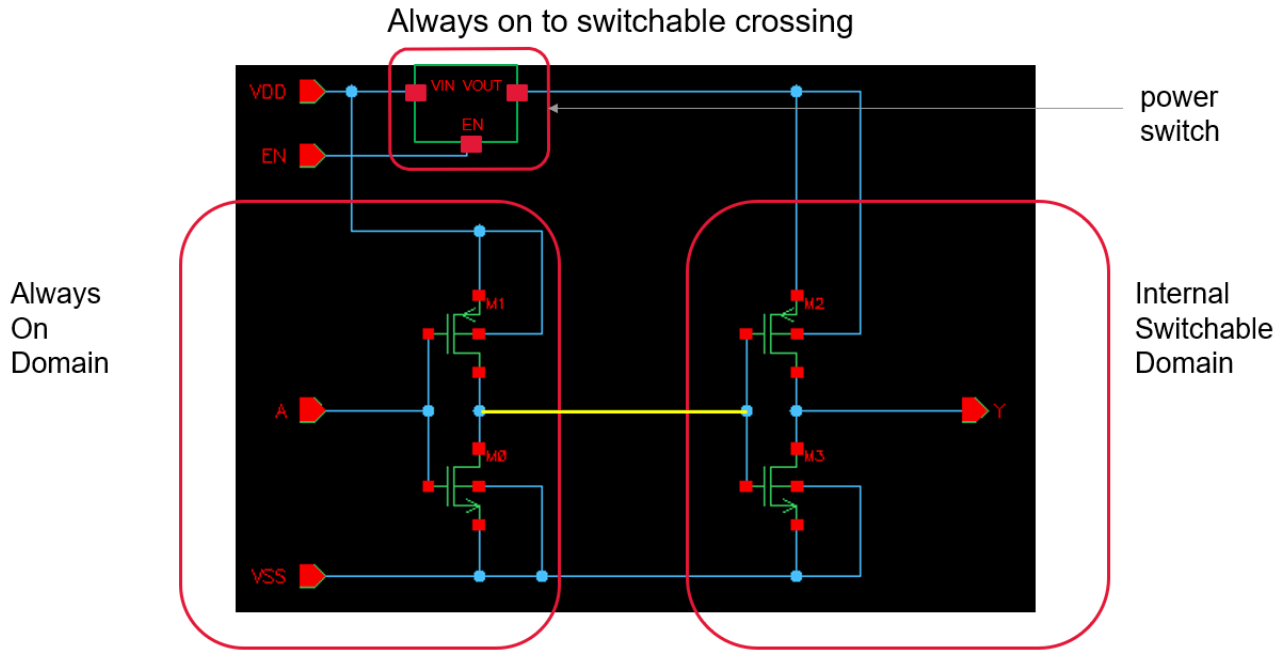


Isolation Checks

- **Missing isolation check**

  Reports all domain crossing involving at least one switchable domain and another switchable or always on domain without isolation. The driver supply could be MOS drain

terminals, standard cell output, or output macro domain ports. Similarly, load could be MOS gate terminals, standard cell input, or macro domain ports.

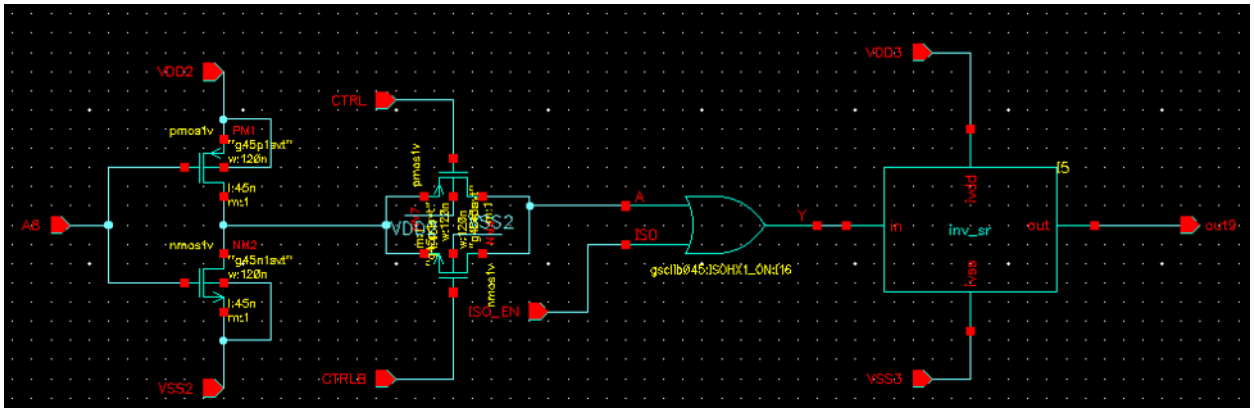


- **Incompatible isolation check**

  Checks the isolation cell for compatibility with the enable pin or supply connection while ensuring that the power shutoff domain has been isolated from functional logic. If the supply for the driver or receiver node is incompatible, an error is flagged. In various user-defined supply states of externally shut-off conditions, it reports if the voltage exists in all the data connections.

- **Redundant isolation check**

Reports an error if the driver and receiver of an isolation cell are switched on and off simultaneously or they are never turned off. This helps in optimizing the design area and power by indicating the redundant circuit elements.

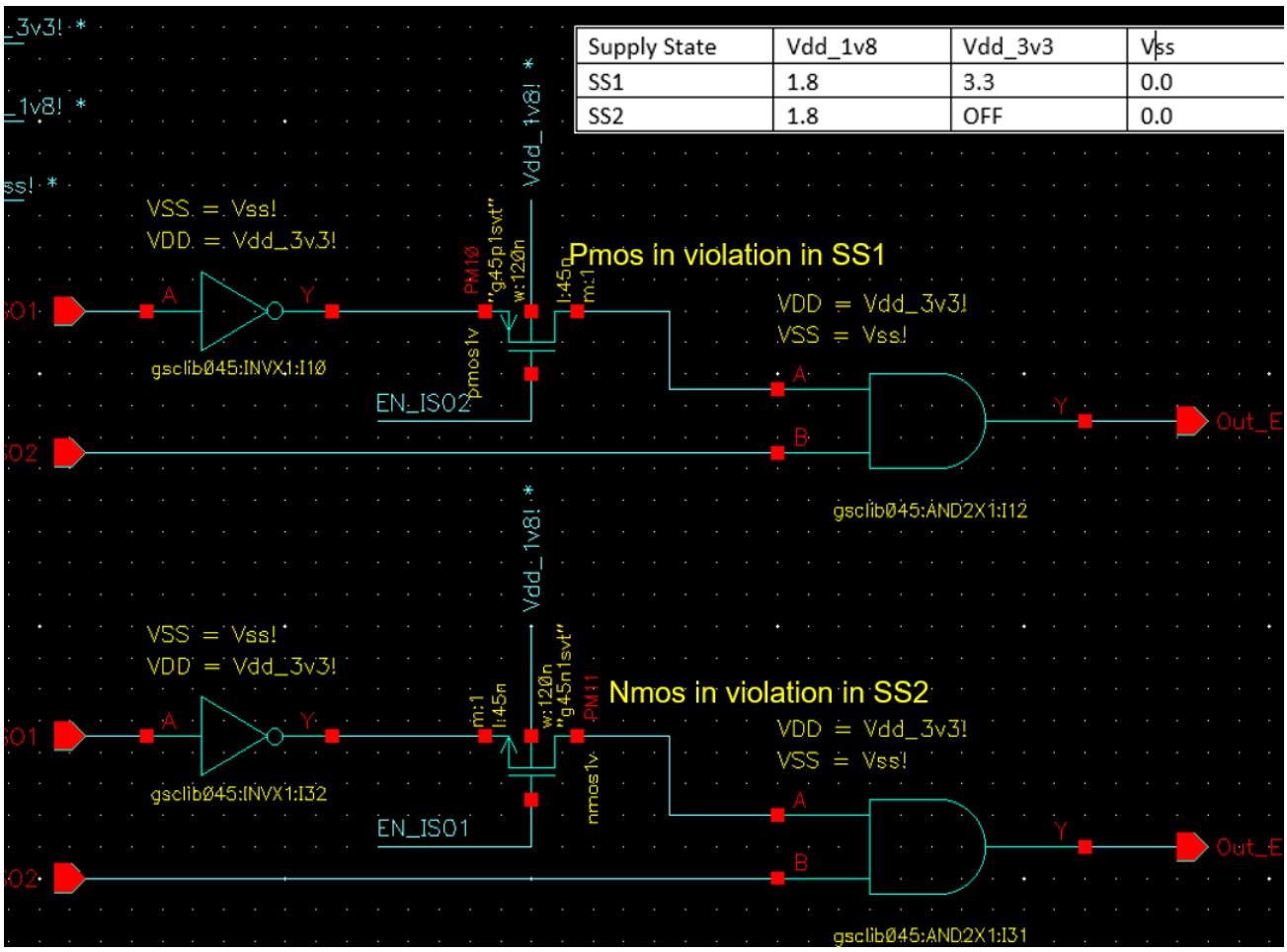


Bulk Checks

The incompatible bulk check flags transistors, which have the bulk terminal-related supply incompatible with the related supply of its source or drain terminal.

- For a P-type transistor, a violation is reported in the following scenarios:
  - The voltage for the related bulk supply net is less than the voltage for the related source or drain supply net.
  - The bulk node is OFF compared to either the source or drain terminal that are fully ON in a particular supply state.
- For an N-type transistor, a violation is reported in the following scenarios:
  - The voltage for the related bulk supply net is more than the voltage for the related source or drain supply net.

- The bulk node is fully ON compared to the source or drain terminal that are OFF in a particular supply state.



Running Checks

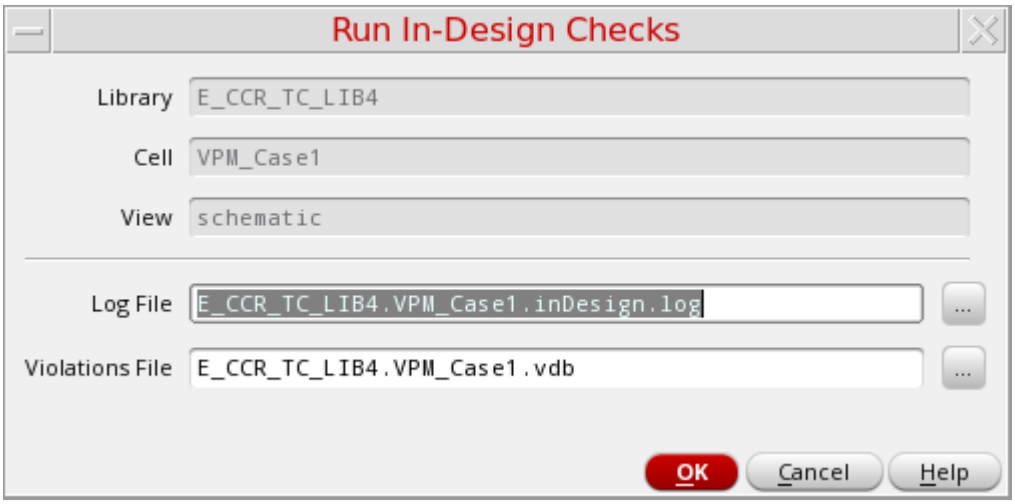
This topic explains running certain checks while designing, loading violations, and analyzing the reported errors in the Annotation Browser. You can browse and review the errors. The cross-highlighting enables to view the errors in the schematic.

- Checking a Design
- Loading the Violations Database
- Filtering Violations

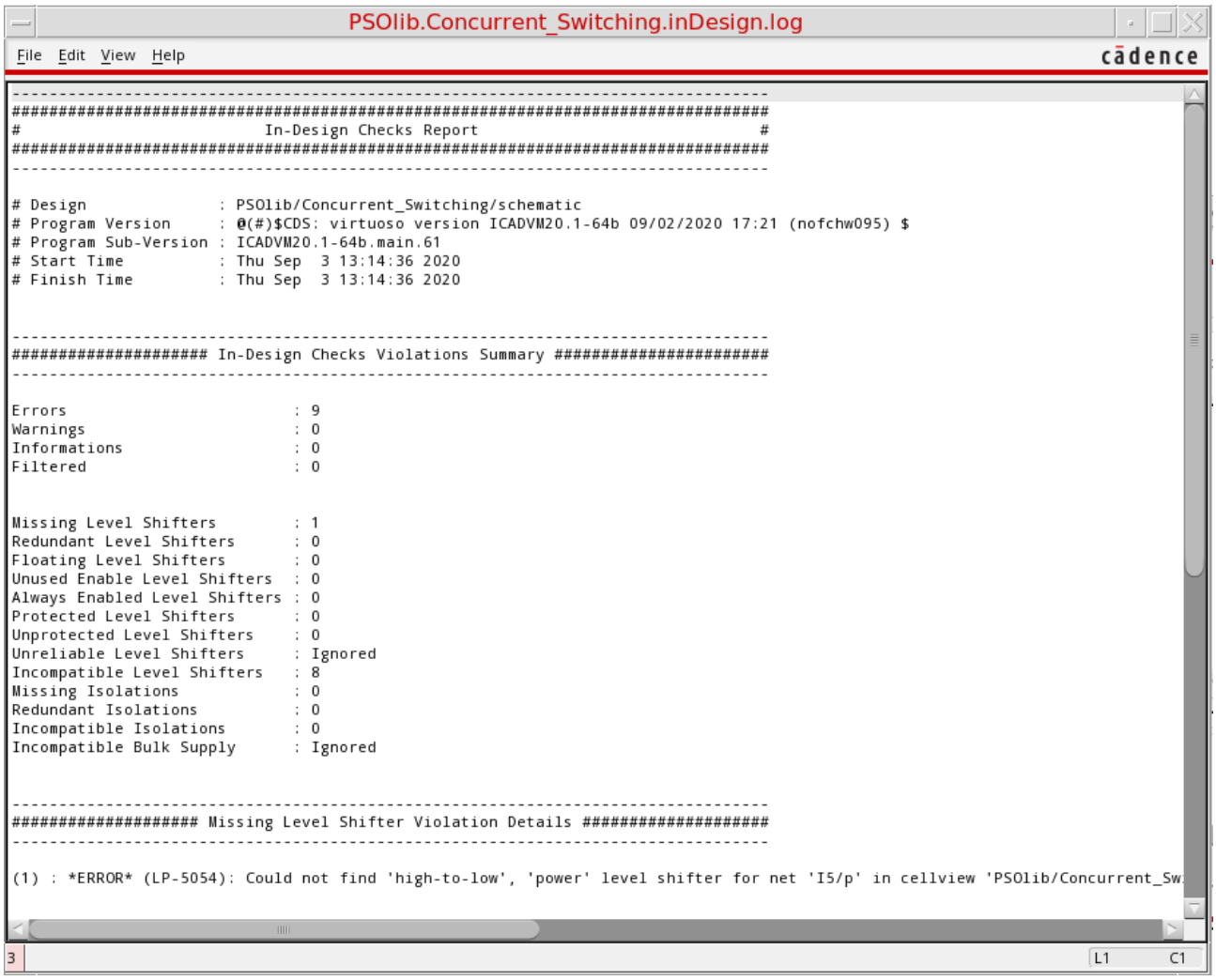
Checking a Design

To run the In-Design Checks, perform the following steps:

1. Load the setup in the Power Manager Setup form.
2. Choose *Power Manager – Run In-Design Checks*.



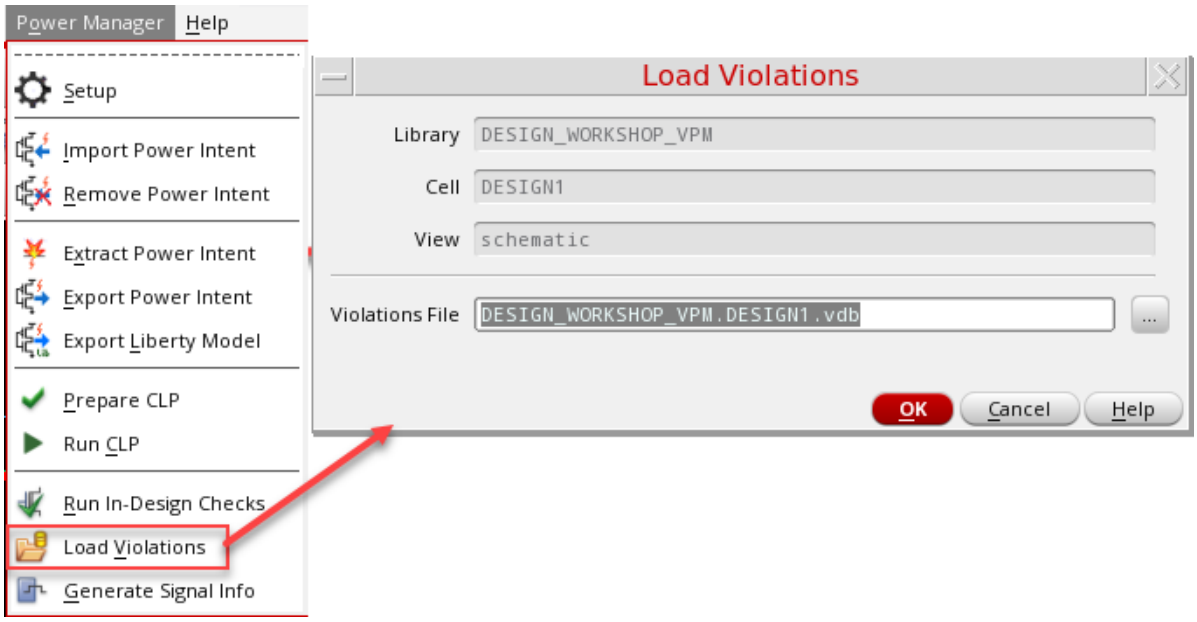
An `inDesign.log` file, such as the one shown here, appears.



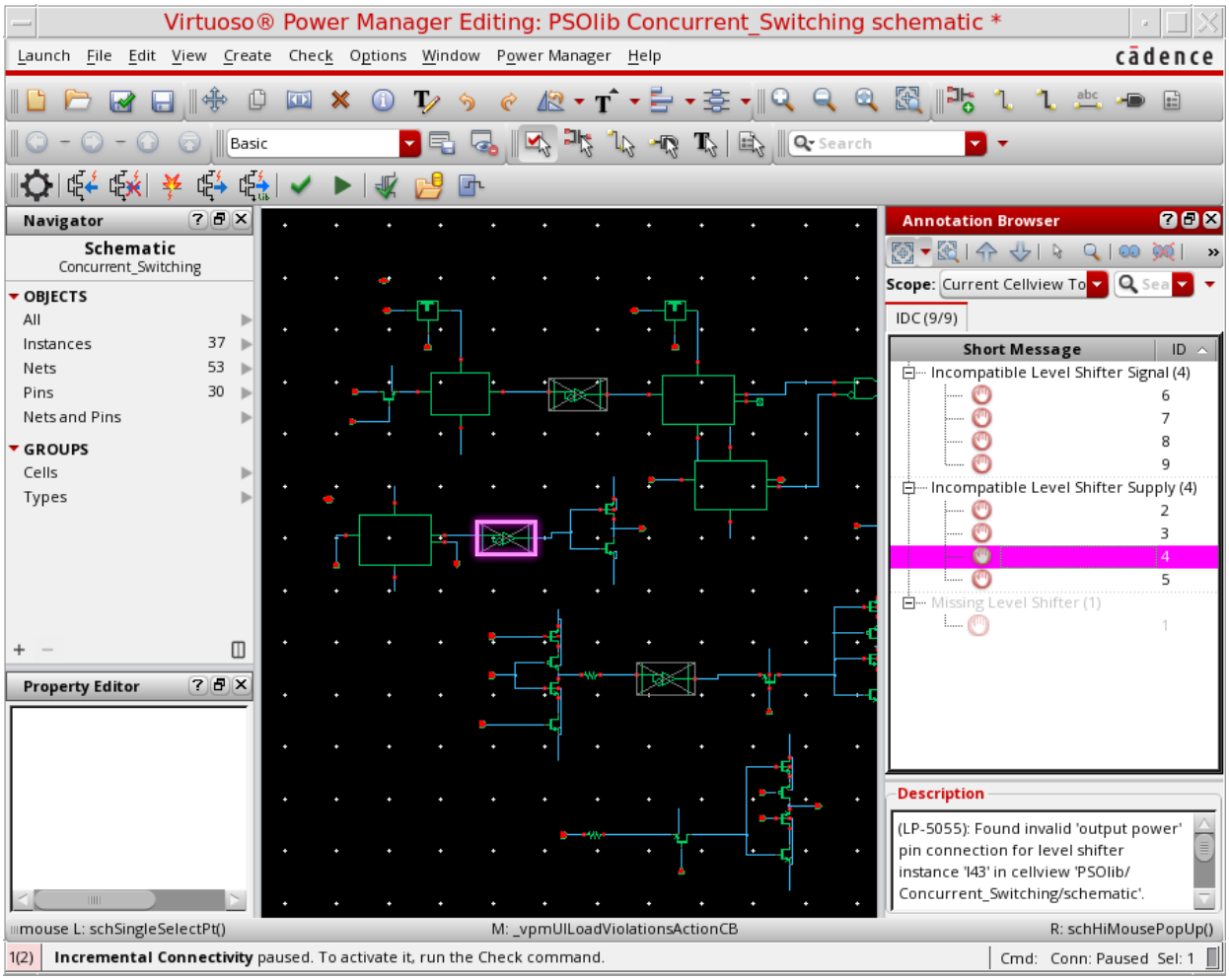
Loading the Violations Database

The errors generated are cross-probed to the accurate schematic location to enable the editing and correction. To load the violations database, perform the following steps:

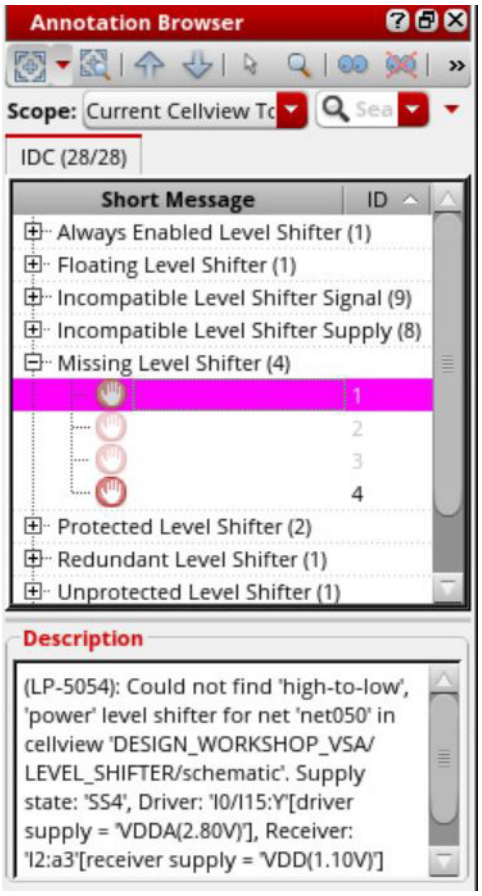
1. Choose *Power Manager – Load Violations*.



2. Specify the file name and click *OK*. The Annotation Browser displays the violations in the design.



3. Specify the scope of the violations, which is based on the cellview hierarchy, to be displayed in the Annotation Browser.



Filtering Violations

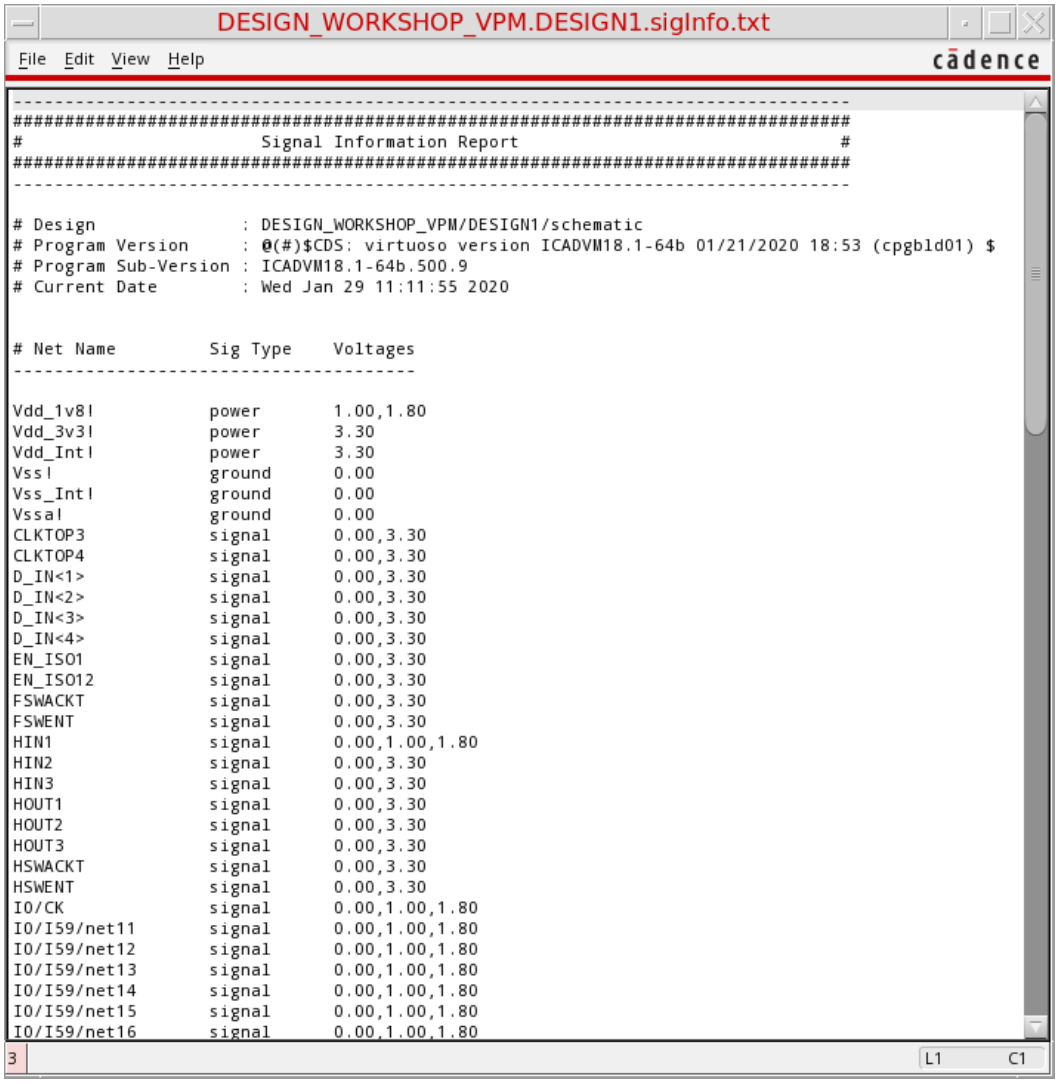
You can add the filter patterns to waive off violations in the *Filters* field on the Runs In-Design Checks tab of the Power Manager Setup form. These indicate the permissive deviations in the design. The tool filters the errors messages or violations matching the pattern specified in the *Filters* field. The Annotation Browser does not display the filtered violations. All matching violations are reported as the filtered messages in the In-Design Checks Report. See In-Design Checks.

Generating Signal Information

You can assign the voltage information to each design net based on the possible voltage values defined in the Signal Information Report that the net can have under multiple design conditions.

To generate the signal information report:

1. Load the setup file. The setup file is needed for identifying the design elements and registering voltage values for different nets.
2. Choose *Power Manager – Generate Signal Info*. Alternatively, use vpmGenerateSigInfo. The Signal Information Report is created.



6

Exporting Power Intent of a Design

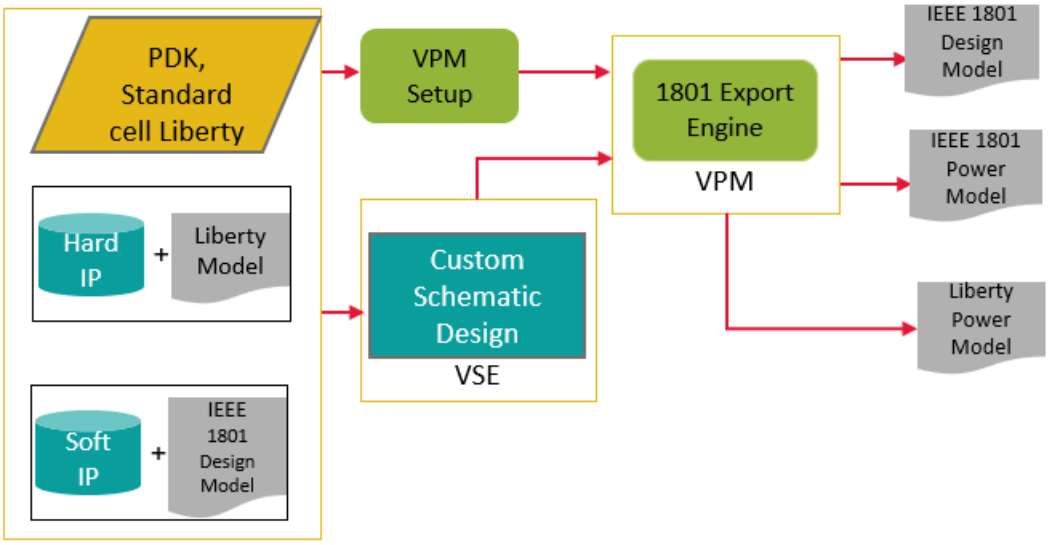
The power intent of the design specified in the 1801 file acts as a design source along with the logical intent (synthesized Verilog netlist). This collection of source input files is utilized by different tools, including the formal verification tools. The 1801 information is expected to successively refine at various design stages, one of which is the case where the design information changes (ECO). Here, a 1801 file incorporating all the design changes is required to be regenerated to have a logical equivalence with the updated design schematic.

The export flow enables you to extract the design connectivity from the schematic and export the low power design intent to a 1801 file. This helps in adding the updated content in incremental stages of IP authoring, which can finally be verified for correctness using CLP along with a VerilogPG netlist.

The chapter includes the following sections:

Export Flow

The illustration below gives an overview of the export flow.



Export Flow in Virtuoso Power Manager

Here are the key stages of the export flow.

- Extraction of design
  - Partitioning of the design
  - Identification of PG net types, standard, special and other cell types
  - Propagation of power information to sub blocks down the hierarchy
  - Creation of power domains and power modes
- Creation and export of Design Model of the design
- Creation and export of Power Model of the design
- Creation and export of Liberty Macro Model of the design

Extracting the Power Intent from a Design

If your design includes objects that describe the power intent for the design, you can extract the power intent, which can further be exported to a 1801 file, to be used further in design flow.

To extract power intent from a design:

1. Open the design in Power Manager.
2. Prepare the setup for automatic extraction. For more details about how to prepare the setup, refer to Specifying the Setup. Load the setup from Power Manager toolbar or menu.
3. On the Power Manager toolbar or menu, click *Extract from Design*.

   The Extract Power Intent form appears.

The library, cell, and view name list of the currently open cellview are shown in the form. The fields specify the information of the cellview, which is non-editable, being extracted.

For a pure schematic-based design, Power Manager always extracts a flat (design model, power model, or macro model) 1801. If there are hierarchical blocks in the design that have their own associated 1801 file, the tool does not extract details of that block and generates a hierarchical 1801, instead by integrating the 1801 file of the lower-level blocks. During the extraction of design schematic the following steps are performed:

Identification of Design Objects

The design extraction involves the identification of the essential design objects, which are required for building a correct power intent. These include:

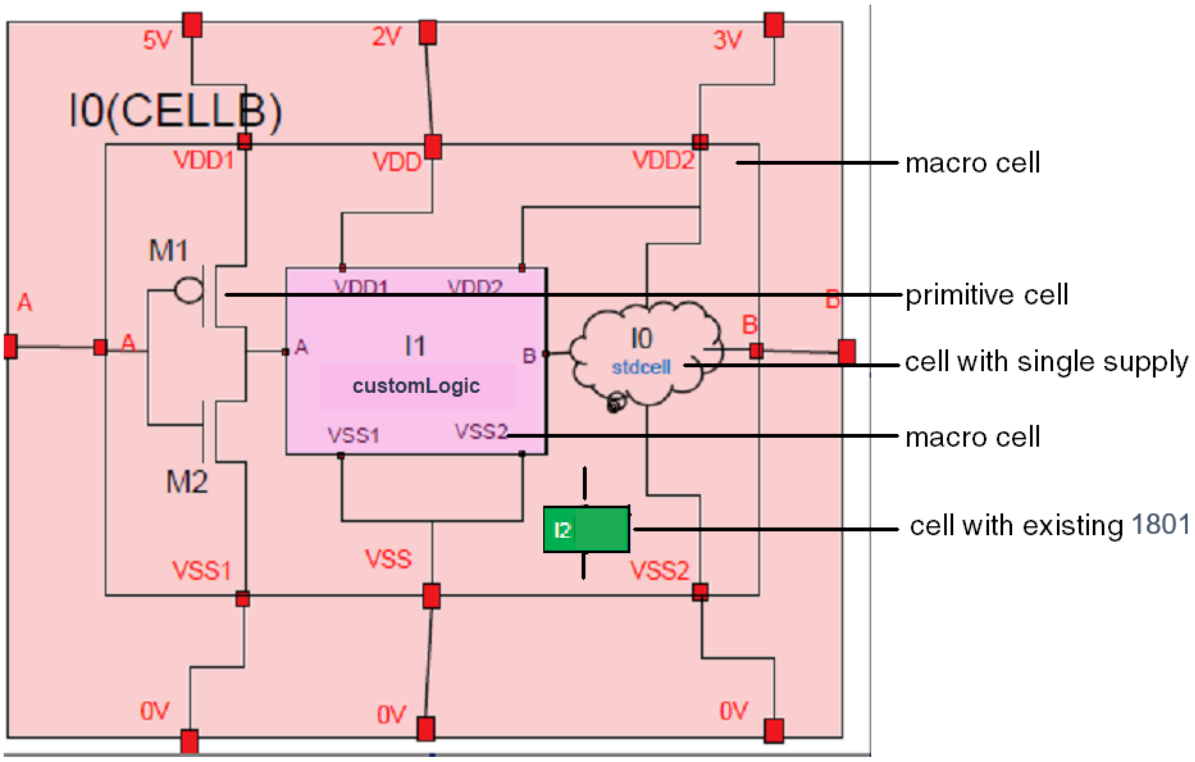
- Identification of power and ground nets associated with each cell in the design hierarchy by using the name-based registration in the setup file.

- Identification of the standard and special cells that have associated Liberty models or a 1801 special cell definition file at the path specified in the setup. When found, the tool identifies the cells defined in the 1801 file as standard or special cells.

For details, refer to Specifying the Setup.

Partitioning of the Design

A design can contain different types of cells. For example, the following example design contains various types of blocks.



During extraction, the extractor handles each type of cell differently. For example, if a cell is already bound with a 1801 file that was previously imported in the schematic of the cell, the extractor refers to the1801file instead of re-extracting the power information of the cellview.

Similarly, for all multi-supply cells that contain primitive cell instances that were identified during the partitioning stage, the extractor extracts the cellview-level power domain information and stitches it to the top-level domains. Instances of single supply cells are considered as standard cells and their power attributes are derived from their corresponding Liberty model and stitched to the top-level power domains. Therefore, before extracting power intent, the tool needs to identify different types of cells and blocks in the design.

During partitioning, the Schematic Editor traverses through the design and identifies the following types of cells:

Cell Type	Description
Primitive cells	A cell that meets the following criteria: ■ Does not contain any instances ■ Does not have any power pin or ground pin Examples: PMOS, NMOS, RES
Passive cells	A cell that meets the following criteria: ■ Contains only primitive instances ■ Does not have any power pin or ground pin Examples: resistor bank or capacitor bank
Low power special cells	A cell that is registered as a special low power cell. Examples: isolation cell, level shifter, power switch
Single supply voltage cells	A cell that meets the following criteria: ■ Contains zero or more primitive cell or passive cell instances ■ Has one power pin and one ground pin Example: single-rail cell
Multiple supply voltage cells	A cell that meets the following criteria: ■ Contains zero or more primitive cell or passive cell instances ■ Has more than one power pin and one or more ground pins Example: multi-rail cell

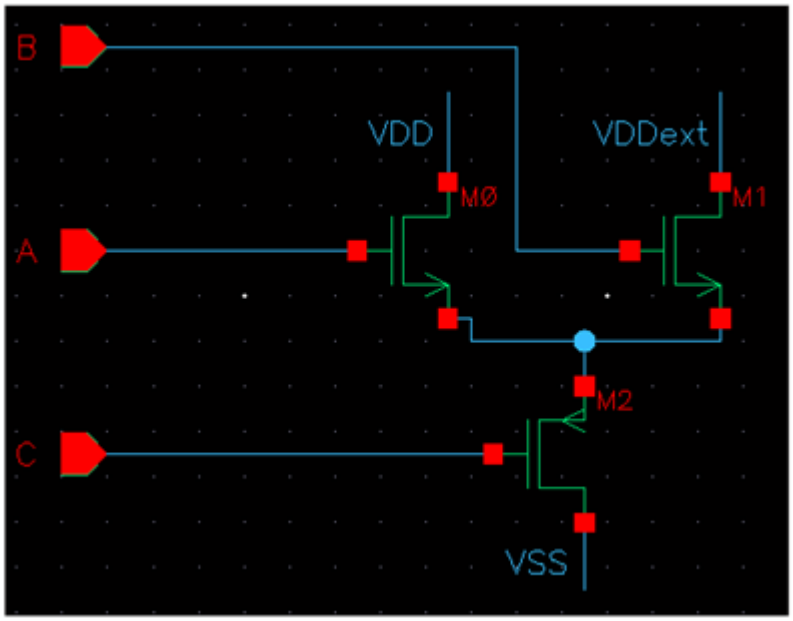
Cell Type	Description
1801 cells	A cell that contains the existing 1801 model. The power intent that has been imported by using the *File – Import Power Intent* command is being applied to this cell. For more details on importing power intent, refer to Importing the Power Intent of a Design.
Macro cells	A cell that meets any one of the following criteria: ■ Contains one or more instances of primitive cell or passive cell and one or more instances of single supply cell, multiple supply voltage cell, or 1801 cell ■ Contains one or more instances of primitive cell or passive cell and has more than one power pin and one or more ground pin ■ Has more than one power pin and one or more ground pin, but does not contain any cell, that is, an empty cell with multiple supply voltages
Hierarchy cells	A cell that meets the following criteria: ■ Contains instances of single supply cell, multiple supply voltage cell, macro cell, or 1801 cell ■ Does not contain any primitive cell or passive cell

After the design partitioning, the next step is the design elaboration. The design elaboration involves the creation of an Embedded Module Hierarchy (EMH), which is a model for storing the hierarchical design data supported at the open access database level. For Power Manager, this open access cellview is called the power view. For details about the power view, refer to Power View. The extractor traverses the entire schematic hierarchy to achieve the following:

- Builds a data structure that contains the OA equivalent objects from the database objects encountered. The structure includes a top-level design, instances, nets, terms, instTerms and net connectivity.
- Utilizes all the instances of the cells that have Liberty models, associated power views, and associated 1801 model.
- Extracts power intent by tracing the connectivity. The connectivity tracing is done considering the block domain of open access cellview, that is, the flat representation of the entire schematic hierarchy in EMH.
- Identifies power nets and ground nets and their equivalent power or ground nets by the sigType in the schematic or the name-based registration in the setup.
- Identifies switchable or non-switchable power nets or ground nets. This is the output from switch supply pin of switch cell instances (Power Switch).
- Pairs the power nets and ground nets. PG net pairing is done by looking at the related PG pins of standard cell instances, special cell instances, macro cell instances. In addition, pairing is also done by tracing a path between the top-block power and ground nets through the source-drain terminals of transistors, shorts, and connector devices in the top block.
- Models various states where every single supply each state is assigned a specific voltage or the state of supply sets, that is, collection of supplies instead of individual supplies by using PST for exporting design and power models.
- Assigns instances to the power domains.
- Creates power states and the power modes. Also, assigns power domains to these power modes.
- Identifies boundary ports and assigns these ports to the power domains. This identification is done based on tracing the connectivity through transistors and two terminal devices. Related power and ground nets of ports are derived by analyzing the related supplies of the driver or receiver instTerms in the top block. While identifying ports for creating power domains, it is checked whether the port could be related to more than one power domain by virtue of its connections to one or more instances that may belong to multiple power domains. In such a situation, the port is not associated with any power domain and the warning messages are issued depending on the situation. There are three scenarios where the ports are not related to any of the power domains due to ambiguity in deciding the power domain:

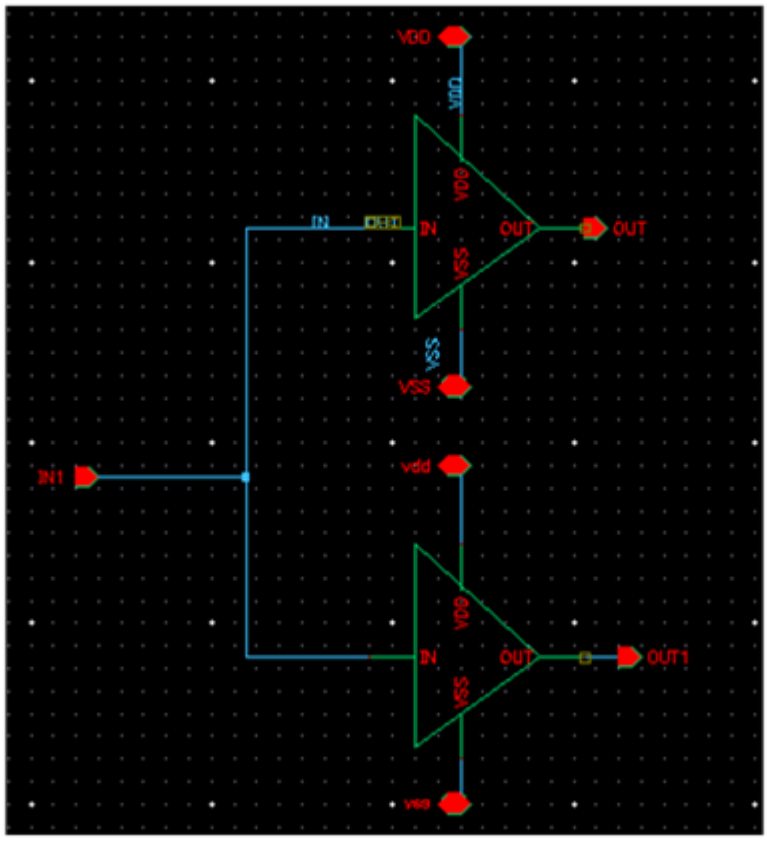
  Case 1: In this scenario, port C can be related to either of the two supply sets (`SS__VDD__VSS` and `SS__VDDext__VSS`) due to its connectivity to `M2`, which falls in

the path of `VDD`, `VSS` as well as `ExtVDD`, `VSS`. Therefore, port C is not related to any supply set because there is ambiguity related to its power domain.



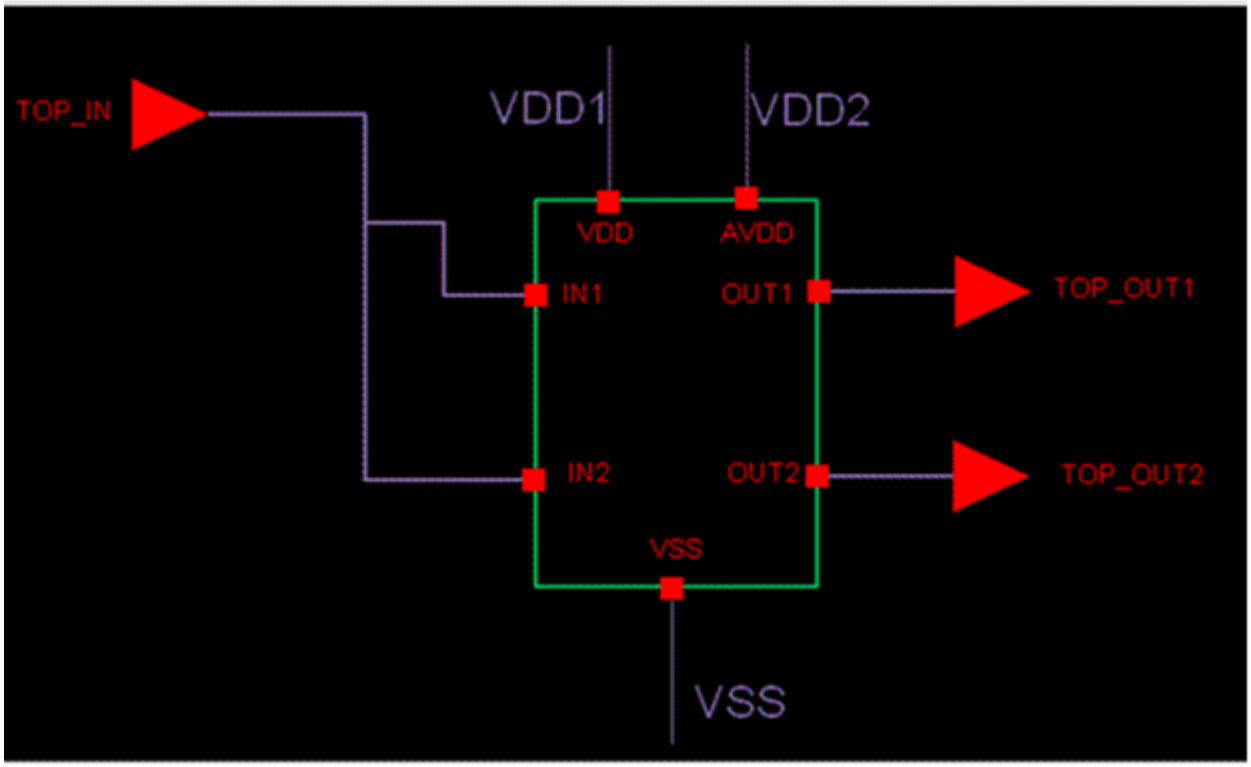
Case 2: In this scenario, the port IN1 is driving two inverter instances that belong to different supply sets and so it can be assigned either to `SS__VDD__VSS` or

SS__vdd__vss. Therefore, port IN1 is not related to any supply set because there is ambiguity related to its power domain.



Case 3: In this scenario, the port TOP_IN drives two pins, IN1 and IN2, of a hierarchical instance. Inside the instances (not shown in the image), IN1 and IN2 are separately driving two inverters, which are in two separate supply sets SS__VDD1__VSS and SS__VDD2__VSS, respectively. The port TOP_IN can be assigned either

SS__VDD1__VSS or SS__VDD2__VSS, therefore, it is not related to any supply set due to ambiguity in its power domain.



In such scenarios, you can provide the information of data ports to explicitly specify their supply set, forcing the tool to associate the data ports to the desired power domain. For details on how to register this information, refer to Supply Set and Power Domain Registration.

While extracting power intent, Power Manager gives precedence to the explicit registration of the port attributes and traces connectivity through transistors and the two-terminal devices. Each data terminal that can be traced to a power terminal is created as a boundary port for the corresponding power domain. If a data terminal is connected to more than one power terminal, it is attached to the power domain as specified in the port attribute registered in the setup for that data terminal. If the port attribute is not registered in the setup, the tool would associate the data terminal to all possible power domains found by connectivity tracing.

Note: While extracting power intent, the tool checks each macro cell and the top cell to find ports that are not related to any supply terminal. The tool does not assign such ports to any power domain and displays a warning message for each cell to report such ports in that cell, as shown below:

```
\w *WARNING* (LP-3034): Could not determine driver/receiver supplies
for logic port 'enps_3v3_i, dll_amux_ao, dll_ibias_10u_ai' in the
Liberty or setup files. Ensure that the required supplies are defined
in the files.
```

- Derives domain shutoff conditions by backtracing the enable pins of switch cell instances in the top block to the top block ports and combining those ports in suitable expression as per the design.
- Integrates sub blocks that have their own 1801files.
  - If the 1801 block has a power intent specified as a Liberty macro model, it identifies the top-level domain that maps to the macro-level domain and creates a domain mapping. PG net voltages in macro cells are assigned from the top net voltages. This might lead to a conflict if the net voltage specified in the macro model are of different voltage levels as compared to the voltage levels of the top net voltage. This conflict can be noted at the verification stage of the power intent.
  - If the 1801 block has power intent of the design model type, Power Manager collates all the power domains in the 1801 block hierarchy and integrates the 1801 cell in the same way as for a block with the macro model type.

  For more details about how to import an existing 1801 file for a sub block, refer to Importing the Power Intent of a Design.
- Identifies low power special cells and creates power switch rules, isolation rules, and level shifter rules, as required. Enable conditions are derived by backtracing the enable pins of special cells.

Note: If a level shifter cell contains an enable pin, it is identified as both an isolation cell and as a level shifter cell. Such cells are also referred as combo cells. In this case, while extracting power intent, the tool defines it as a level shifter cell as well as an isolation cell. In addition, corresponding level shifter and isolation rules are defined.

Creation of Power Domains

While extracting power intent, Power Manager automatically identifies the power nets and ground nets in the design as described below. Further, for each power net, it finds the ground net associated with it and for each unique pair of power net and ground net, creates a power domain.

Identifying Switchable and Non-Switchable Power Nets and Ground Nets

- A net is identified as a non-switchable power net if either of the following conditions are met:

- the sigType property of the net is power
- the net name is registered as a power net
- the net is not an output of a switch cell and cannot be hierarchically traced to an output of a switch cell.

- A net is identified as a switchable power net if either of the following conditions are met:
  - the net is an output of a head-switch cell
  - the net can be hierarchically traced to an output of a head-switch cell
- A net is identified as a non-switchable ground net if:
  - the sigType of the net is ground.
  - the net name is registered as a ground net.
  - the net is not an output of a switch cell and cannot be hierarchically traced to an output of a switch cell.
- A net is identified as a switchable ground net if either of the following conditions are met:
  - the net is an output of a foot-switch cell
  - the net can be hierarchically traced to an output of a foot-switch cell

Note: If you have not registered any names for power nets and none of the nets in the design have their signal type set to power, then Power Manager does not extract any power domains.

After identifying the power nets and ground nets, the unique power net/ground net pairs (Supply sets) are identified. For this, the tool traces the paths from a power net to a ground net. The tool also considers the power net/ground net pairs registered in the setup.

Supply sets can be defined in the setup for the following scenarios.

- The primary supply set for the power domains mentioned in the setup.
- The power and ground nets that cannot be paired by traversing the design hierarchy or parsing the Liberty in case of a Liberty cell.

Supply set identification can be done explicitly in a setup template or by automatic identification.

If you want to partition the design for assigning a particular instance to some power domain, provide the list of domain, instances, and primary supply set. For more information refer to Supply Set and Power Domain Registration.

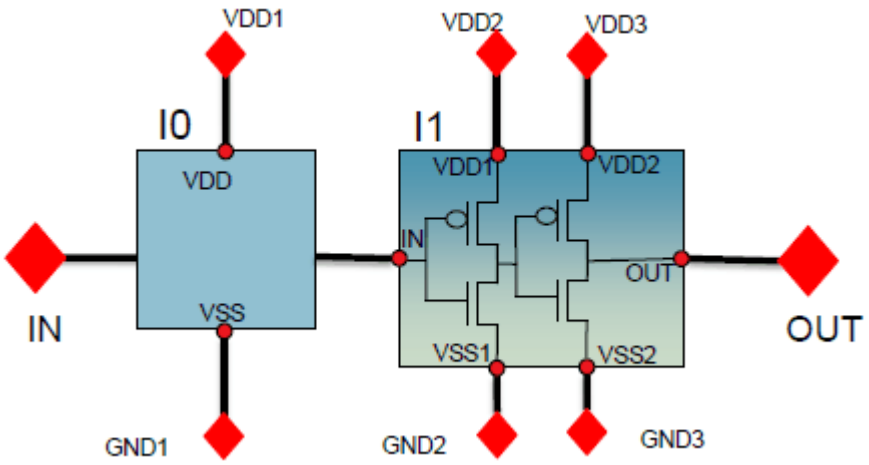
If power domains and supply set information is provided in the setup, the following is the sequence of tasks being done:

- `create_power_domain` commands are created in the exported 1801 file based on the number of power domains.
- Primary supply set are referred from the setup.
- Extra supplies that need not to be a part of setup are identified by the tool.
- `connect_supply_net` command is generated for supply nets that are a part of extra supplies.
- For any other instance that is not a part of any power domain remains in the default domain.
- If there is no power domain assignment, a single power domain is created and all the instances are connected by using `connect_supply_net`.

The following examples show how power domains are created:

Example 1:

Consider the following design that does not have the user-defined supply sets and power domains specified in the setup:

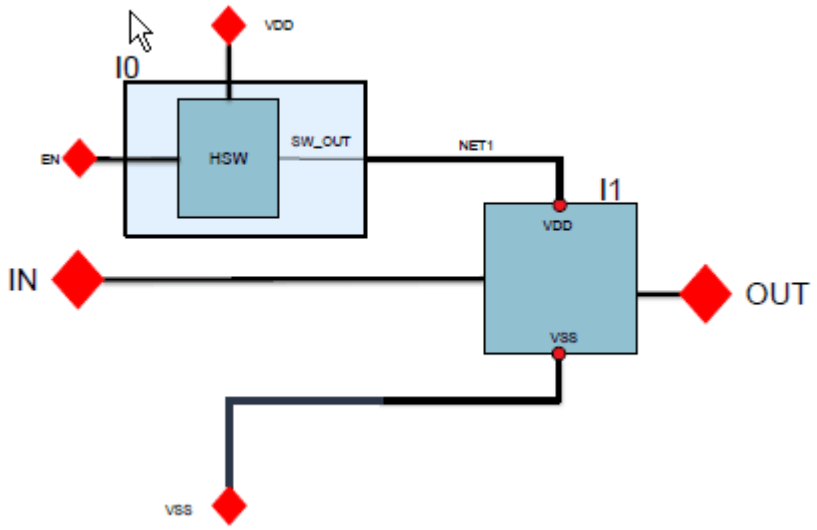


In this case, the tool traverses though the design and creates the following power domains:

- `SS__VDD1__GND1`
- `SS__VDD2__GND2`
- `SS__VDD3__GND3`

Example 2:

Now, consider the following design that contains a head-switch.



In this case, the tool creates one default power domain and two supply sets:

- `pd__VDD__VSS`
- `SS_VDD_VSS`
- `SS_NET1_VSS`

Example 3:

If a primitive instance has one netSet property each for a power and ground supply, it is added to the power domain created for that power and ground supply pair even if the instance does not have any power/ground pins in the symbol/schematic view. If a power domain does not exist for that power and ground supply pair, a new power domain is created and the primitive instance is added to it. However, if the primitive instance has more than one netSet property for power or ground, it does not contribute towards the creation of a power domain and a warning message is displayed in the power intent extraction log.

Power View

Power view is an OA view generated corresponding to each cell that is being extracted to export the power intent. The power view stores the following information:

- Set up information of the cellview prior to extraction

- Extracted and/or imported power intent information
- The design information derived from the schematic design (EMH and native OA objects)
- Internal data structures

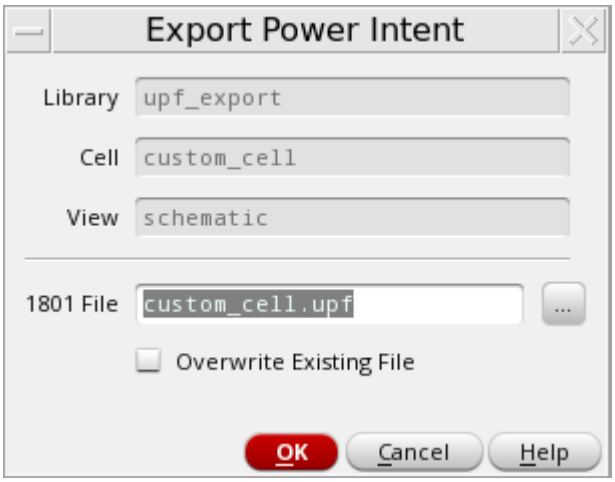
The power view is co-managed with the schematic view of the design. The view is automatically updated upon extraction when the setup or design changes. There is no need to explicitly load the setup for each successive extraction run until the power view exists for a cell and there is no change in the setup information of the design. The power view by default gets created in the design library within the design cell, adjacent to the design view/ schematic. If the parent library is non-editable, lpDBGlobalSearchLibs and lpDBGlobalSearchViews can be used to save the power view in an editable user-defined library. The power view is also automatically purged when the associated schematic cellview is purged for reasons, such as on closing the schematic or calling the `dbPurge`/`dbClose` SKILL function.

Exporting 1801 Design Model

You can export the power intent specified for your design to a 1801 design model file. To export power intent from a design:

1. Open the design in Power Manager.
2. Prepare the setup for automatic extraction. For more details about how to prepare the setup, refer to Setup Preparation for Automatic Extraction of Power Intent. Load the setup from Power Manager toolbar/menu.
3. Click *Power Manager – Extract from Design*.
4. Click *Power Manager – Export from Design*.

The Export Power Intent form appears.



The library, cell, and view names of the cell that is currently open are displayed by default in the form. In the *1801 File* field, specify the path and the name of file for exporting the power intent and click OK. Select the *Overwrite Existing File* check box to overwrite the file.

The vpmExportPowerIntent SKILL function has been provided to export the 1801 Design Model information for a cellview.

In 1801 design model, the power intent of the sub-blocks is exported along with power intent of the top design. For a pure schematic-based design, the Power Manager always extracts a flat 1801.

An example of an exported 1801 design model file is shown below:

```
set_design_attributes -attribute top_ports_have_anon_supply 0
##### Supply Ports (Section 1)#####################################################
create_supply_port VDD       -direction inout
create_supply_port VDDA      -direction inout
create_supply_port VDDSW     -direction inout
create_supply_port VSS       -direction inout

##### Supply Nets (Section 2)######################################################
create_supply_net VDD   -resolve parallel
create_supply_net VDDA  -resolve parallel
create_supply_net VDDSW -resolve parallel
create_supply_net VSS   -resolve parallel

##### Supply Net Connections (Section 3)###########################################
connect_supply_net VDD    -ports { VDD }
```

```
connect_supply_net VDDA   -ports { VDDA }
connect_supply_net VDDSW  -ports { I3/I0/VDD }
connect_supply_net VSS    -ports { VSS }

##### Supply Sets (Section 4)#################################################
create_supply_set SS_VDDA_VSS  -function { power VDDA } -function { ground VSS }
create_supply_set SS_VDDSW_VSS -function { power VDDSW } -function { ground VSS }
create_supply_set SS_VDD_VSS   -function { power VDD } -function { ground VSS }

##### Power Domains (Section 5)###############################################
create_power_domain PD_TOP -include_scope -supply { primary SS_VDD_VSS } \
-supply { extra_supplies_1 SS_VDDSW_VSS }
create_power_domain PD_LS -elements { I0 } -supply { primary SS_VDDA_VSS } \
-supply { extra_supplies_1 SS_VDD_VSS }

###### Standard Cells (Section 6)#############################################
connect_supply_net VDD   -ports { I0/I20/I0/VDD }
connect_supply_net VDDSW -ports { I3/I1/VDD }
###### Special Cells #########################################
connect_supply_net VDD   -ports { I0/I22/ExtVDD }
connect_supply_net VDDSW -ports { I3/I0/VDD }
###### Macro Cells ###########################################
connect_supply_net VDD   -ports { I2/VDD_ext }
connect_supply_net VSS   -ports { I2/VSS_ext }

##### Port Attributes (Section 7)#############################################
set_port_attributes  -port en_iso  -receiver_supply SS_VDD_VSS
set_port_attributes  -port in1  -receiver_supply SS_VDD_VSS
######### Isolation Rules ###################################
set_isolation PD_TOP_ISO_0 -domain PD_TOP -elements { out_iso } -
isolation_supply_set  SS_VDD_VSS -isolation_signal en_iso -isolation_sense high -
location self -clamp_value 0


######### Level Shifter Rules (Section 8)############
set_level_shifter PD_LS_ls_0 -domain PD_LS -elements {I0/out_ls } -
input_supply_set SS_VDDA_VSS \ -output_supply_set SS_VDD_VSS -location self -rule
both
##### Switch Rules ##################
create_power_switch I3_I0 -output_supply_port { VDD VDDSW } -input_supply_port {
ExtVDD VDD } \
-control_port { PSO psw_en } -on_state { on ExtVDD {!PSO} } -off_state { off PSO }
-domain PD_TOP
```

```
#####   Supply Port States (Section 9) ##########################################
add_port_state I3/I0/VDD   -state { V110 1.1 }  -state { OFF off }
add_port_state VDD       -state { V110 1.1 }  -state { OFF off }
add_port_state VDDA      -state { V130 1.3 }  -state { OFF off }
add_port_state VSS       -state { OFF 0 }

#####   Power State Table (Section 10) ##########################################
create_pst              top_pst -supplies      [list I3/I0/VDD  VDD   VDDA  VSS   ]
add_pst_state State_1    -pst top_pst -state    {     V110       V110  V130  OFF   }
add_pst_state State_2    -pst top_pst -state    {     OFF        V110  V130  OFF   }
add_pst_state State_3    -pst top_pst -state    {     OFF        OFF   OFF   OFF   }
```

The following points explain the different sections of the exported 1801 design model:

- **Section1**: The `create_supply_port` command defines a supply port in the scope of the power domain. These are created for all the top-level supply ports identified.
- **Section2**: The `create_supply_net` command creates a supply net in the scope of the power domain. These are created for all the top-level supply nets identified.
- **Section3**: The `connect_supply_net` command connects a supply net to the specified ports. These commands are for handling the interface connectivity. The commands are for both the top level as well as for hierarchical interface connectivity. This resolves the connectivity of PG nets from down the hierarchy to the top level PG nets for different blocks. Section6 also performs the same task.
- **Section4**: The `create_supply_set` command creates the supply set name within the current scope. This defines the primary and extra supply sets or available supplies associated with the different power domains.
- **Section5**: The `create_power_domain` command defines a power domain and the set of instances that are in the extent of the power domain. It may also specify whether the power domain can be partitioned further by the subsequent commands.
- **Section6**: The `connect_supply_net` command defines the standard, special, and macro cells.
- **Section7**: The `set_port_attributes` command specifies the information associated with data ports of instances. The attributes of this command identify a port's related supplies (driver or receiver) and the boundary of a power domain.
- **Section8**: These commands define the low power strategy (Level Shifters, Isolation, and Power Switch) adopted for the ports on the interface of a power domain. This is required to correct for voltage differences between the driving and receiving supplies of a port or to ensure correct electrical and logical functionality when domains are in different power states.

The correct strategy adopted is also based on the cell type identified while reading the schematic based on their Liberty model or special cell definition file registered in the setup. For details, refer to Supply Set and Power Domain Registration.

- **Section9**: The `add_port_state` command adds the state information to a supply port. If the voltage values are specified, the supply net state is `FULL_ON` and the voltage value is the single nominal value or within the range of min to max. If the supply net state is off, the voltage value is OFF. For net voltage registration, refer to Device and Cell Registration.
- **Section10**: The `add_pst_state` command defines the name for a specific state of the supply nets defined for the power state table (PST). This command defines system power states of the IP.

The output supply port that is the output of a power switch, output of a LDO, or voltage regulator can be identified by explicit user registration in the setup. In addition, the output supply port can be identified from the Liberty model or the 1801 file, if available for the block. The commands associated with the output port would appear in Section1, Section2, Section9, and Section10.

A hierarchical internal net is not generated for commands related to Section1, Section2, Section3, and Section6. Any block with internal net (not available at the top) should be extracted and 1801 generated. Then, use the 1801 binding for the top block.

For more information on the 1801 commands, refer to *IEEE Standard for Design and Verification of Low-Power Integrated Circuits*.

Exporting 1801 Power Model

You can export the power intent specified for the design to a 1801 power model file. To export power intent from a design,

1. Open the cellview in Power Manager.
2. Prepare the setup for automatic extraction. For more details about how to prepare the setup, refer to Setup for Automatic Extraction of Power Intent.
3. Load the setup from the Power Manager toolbar/menu.
4. On the Power Manager toolbar/menu, click *Extract from Design*.
5. Use vpmExportPowerModel to export the 1801 Power Model information for a cellview.

In the 1801 power model, the power intent of the top design is only exported as the power intent information, with no sub-block power intent details. A power model is used to define the

power intent of a hard IP cell. The command pair `begin_power_model` and `end_power_model` create a definitive boundary for the power intent for the hard IP cell. A model name is created, and it is targeted for a specific macro cell.

The interfaces of the hard IP power model include the top-level power domain for this IP cell and all the supplies of IP. This power domain is also used to specify the system power states at the hard IP level.

An example of an exported 1801 power model file is shown below:

```
upf_version 2.1
##### Power Model Initialization (Section 1)###################################
begin_power_model top_model -for top
##### Supply Ports (Section 2)#################################################
create_supply_port VDD!  -direction inout
create_supply_port VDDA! -direction inout
create_supply_port VSS!  -direction inout
##### Supply Nets (Section 3)##################################################
create_supply_net VDD   -resolve parallel
create_supply_net VDD1  -resolve parallel
create_supply_net VDDA  -resolve parallel
create_supply_net VDDSW -resolve parallel
create_supply_net VSS   -resolve parallel
create_supply_net VSS1  -resolve parallel
connect_supply_net VDD    -ports { VDD }
connect_supply_net VDDA   -ports { VDDA }
connect_supply_net VDDSW  -ports { I3/I0/VDD }
connect_supply_net VSS    -ports { VSS }
##### Supply Sets (Section 4)##################################################
create_supply_set SS_VDDA_VSS     -function { power VDDA } -function { ground VSS }
create_supply_set SS_VDDSW_VSS    -function { power VDDSW } -function { ground VSS
}
create_supply_set SS_VDD_VSS      -function { power VDD } -function { ground VSS }
##### Power Domains (Section 5)################################################
create_power_domain PD_MACRO -elements {.}  \
-supply { primary SS_VDD_VSS } \
-supply { extra_supplies_0 SS_VDD+VDDSW_VSS } \
-supply { extra_supplies_1 SS_VDDA_VSS } \
-supply { extra_supplies_2 SS_VDDSW_VSS }
##### Port Attributes (Section 6)##############################################
set_port_attributes  -port en_iso  -receiver_supply SS_VDD_VSS
set_port_attributes  -port in1  -receiver_supply SS_VDD_VSS
set_port_attributes  -port in2  -receiver_supply SS_VDD_VSS
```

```
set_port_attributes  -port in3  -receiver_supply SS_VDD_VSS
set_port_attributes  -port in4  -receiver_supply SS_VDD+VDDSW_VSS
set_port_attributes  -port out1  -driver_supply SS_VDD_VSS
set_port_attributes  -port out2  -driver_supply SS_VDD_VSS
set_port_attributes  -port out3  -driver_supply SS_VDD_VSS
set_port_attributes  -port out_iso  -driver_supply SS_VDD_VSS
set_port_attributes  -port out_psw  -driver_supply SS_VDD_VSS
set_port_attributes  -port outvdd  -driver_supply SS_VDD_VSS
set_port_attributes  -port psw_en  -receiver_supply SS_VDD_VSS

##### Power States (Section 7)##########################################
add_power_state SS_VDDA_VSS -supply \
                            -state { ON -simstate NORMAL \
                                -supply_expr { power == { FULL_ON 1.300000 } &&
ground == { FULL_ON 0.000000 } }} \
                             -state { OFF -simstate CORRUPT \
                                 -supply_expr { power == OFF && ground == OFF }}

add_power_state SS_VDDSW_VSS -supply \
                            -state { ON -simstate NORMAL \
                                -supply_expr { power == { FULL_ON 1.100000 } &&
ground == { FULL_ON 0.000000 } }} \
                             -state { OFF -simstate CORRUPT \
                                 -supply_expr { power == OFF && ground == OFF }}

add_power_state SS_VDD_VSS -supply \
                            -state { ON -simstate NORMAL \
                                -supply_expr { power == { FULL_ON 1.100000 } &&
ground == { FULL_ON 0.000000 } }} \
                             -state { OFF -simstate CORRUPT \
                                 -supply_expr { power == OFF && ground == OFF }}

add_power_state PD_MACRO -domain  \
                         -state { S0 -logic_expr { SS_VDD_VSS == ON && SS_VDDSW_VSS
== ON && SS_VDDA_VSS == ON }} \
                         -state { S1 -logic_expr { SS_VDD_VSS == ON && SS_VDDSW_VSS
== ON && SS_VDDA_VSS == OFF }} \
                         -state { S2 -logic_expr { SS_VDD_VSS == ON && SS_VDDSW_VSS
== OFF && SS_VDDA_VSS == ON }} \
                         -state { S3 -logic_expr { SS_VDD_VSS == ON && SS_VDDSW_VSS
== OFF && SS_VDDA_VSS == OFF }}
##### Power Model End (Section 8)#######################################
end_power_model
```

The following points explain the different sections of the exported 1801 power model:

- **Section1**: The `begin_power_model` command defines the boundary condition for the power model. This marks the beginning of the power model.
- **Section2**: The `create_supply_port` command defines a supply port in the scope of the power domain. These are created for all the top-level supply ports identified.
- **Section3**: The `create_supply_net` command creates a supply net in the scope of the power domain. These are created for all the top-level supply nets identified.
- **Section4**: The `create_supply_set` command creates the supply set name within the current scope. This section associates the interface supplies to the boundary supply ports or internally generated supplies.
- **Section5**: The `create_power_domain` command defines a power domain and the set of instances that are in the extent of the power domain. It may also specify whether the power domain can be partitioned further by subsequent commands.
- **Section6**: The `set_port_attributes` command specifies the information associated with data ports of instances. The attributes of this command identify a port's related supplies (driver or receiver) and define the boundary of a power domain.
- **Section7**: The `add_power_state` command adds the state information to a supply port. If the voltage values are specified, the supply net state is `FULL_ON` and the voltage value is the single nominal value or within the range of min to max. If the supply net state is off, the voltage value is OFF. For net voltage registration, refer to Supply Nets Registration.
- **Section8**: The `end_power_model` command defines the boundary condition for the power model. The power model ends here.

Exporting Liberty Power Model

You can export the power intent specified for the design as a Liberty power model template file with the non-characterized attributes for `pg_pin` and pin groups of a library or cell, such as `related_power_pin`, `related_ground_pin`, `pg_type`, `direction`, `is_isolated`, `isolation_enable_condition`, `switch_pin`, `pg_function`, `switch_function`, and so on. To export power intent from a design,

1. Open the cellview in the Power Manager.
2. Prepare the setup for automatic extraction. For more details about how to prepare the setup, refer to Setup for Automatic Extraction of Power Intent.
3. Load the setup from the Power Manager toolbar/menu.

4. Click *Power Manager – Extract from Design*.
5. Click *Power Manager – Export Liberty Model*.

   The Export Liberty Model form appears.

The library, cell, and view names of the cellview that is currently open are displayed by default in the form. In the *Liberty File* field, specify the path and the name of the file for exporting the Liberty power model and click *OK*. Select the *Overwrite Existing File* check box to overwrite the file.

Note: vpmExportDotLib has been provided to export the Liberty power model information for a cellview.

Liberty power model is the power intent at the macro model level, for complex design blocks. This can also be referred as a black box model of the power characteristics of a complex design block or a hard IP. Power Manager helps in automatically extracting the power intent from the design schematic and exporting the low power attributes to a Liberty Power Model template. It can be further integrated while extracting the power intent of the top design. The Exported Liberty Power Model, having the PG attributes can be stitched to the baseline Liberty model from an IP characterization tool to generate a complete liberty file for schematic IP ready for verification and implementation.

While creating the Liberty power model for a macro cell, the following tasks are performed:

- Identifies power nets and ground nets and their equivalent power or ground nets.
- Identifies switchable or non-switchable power nets or ground nets.
- Identifies boundary ports for the macro.

- Relates the boundary ports to the related power and ground supply pairs by tracing.
- Identifies low power special cells and the boundary ports associated with boundary ports to print the relevant attributes.
- Backtraces the data path from the control pins of the low power special cells to the macro or design ports for generating enable or shutoff conditions. Refer to Setup for Automatic Extraction of Power Intent for cell registration requirements to enable backtracing.

The Liberty power model also enables creation of analog ports. Analog ports are a list of ports of a macro cell or a design that either drive the analog circuits or are driven by them. In a Liberty power model, a port is considered an analog port in the following cases:

- There is a user override, by defining `portAttribute`, in the setup that declares the pin as analog.
- The pin has a related power or a related ground found by tracing the power and ground path.
- The pin is not a driver or load of one or more instance pins that produce digital output, such as (Liberty or user-defined) standard cell input/output pin, which has a related power and a related ground attribute.
- The pin is not feedthrough and unconnected.

Ports identified as analog ports have the `is_analog` attribute associated with them.

An example of an exported Liberty Power Model file is shown below:

```
#################################################################
/*
    Liberty Power Model Template (Section 1)
    ============================
    library              : upf_export;
    cell                 : top;
    Program Version      : sub-version  IC6.1.8-64b.main.174 ;
*/
##################(Section 2)#####################################################
library(upf_export) {

    date                 : "Wed Mar  6 18:02:38 2019 ";
    comment              : "Generated by Virtuoso Power Manager";
    voltage_unit         : "1V";

    voltage_map( VDD , 1.100000);
    voltage_map( VDDA , 1.300000);
    voltage_map( VDDSW , 1.100000);
```

```
    voltage_map( VSS , 0.000000);
####################(Section 3)#################################################
    cell(top)  {
        switch_cell_type     : fine_grain;
        is_macro_cell        : true;
        pg_pin(VDD)  {
            voltage_name         : VDD;
            pg_type              : primary_power;
            direction            : inout;
        }
        pg_pin(VDDA)  {
            voltage_name         : VDDA;
            pg_type              : primary_power;
            direction            : inout;
        }
        pg_pin(VSS)  {
            voltage_name         : VSS;
            pg_type              : primary_ground;
            direction            : inout;
        }
        pg_pin(VDDSW)  {
            switch_function      : "psw_en";
            voltage_name         : VDDSW;
            pg_function          : VDD;
            pg_type              : internal_power;
            direction            : internal;
        }
#######################################################
        pin(en_iso)  {
            related_ground_pin   : VSS;
            related_power_pin    : VDD;
            direction            : input;
        }
        pin(in1)  {
            related_ground_pin   : VSS;
            related_power_pin    : VDD;
            direction            : input;
        }
        pin(out1)  {
            power_down_function  : "!VDD + VSS";
            related_ground_pin   : VSS;
```

```
            related_power_pin    : VDD;
            direction            : output;
        }
        pin(out_iso)  {
            is_isolated          : true;
            power_down_function  : "!VDD + VSS";
            related_ground_pin   : VSS;
            related_power_pin    : VDD;
            isolation_enable_condition : "en_iso";
            direction            : output;
        }
        pin(out_psw)  {
            power_down_function  : "!VDD + VSS";
            related_ground_pin   : VSS;
            related_power_pin    : VDD;
            direction            : output;
        }
        pin(outvdd)  {
            power_down_function  : "!VDD + VSS";
            related_ground_pin   : VSS;
            related_power_pin    : VDD;
            direction            : output;
        }
        pin(psw_en)  {
            antenna_diode_related_ground_pins : "VSS";
            switch_pin           : true;
            related_ground_pin   : VSS;
            related_power_pin    : VDD;
            direction            : input;
        }
    } /* end of cell top */

} /* end of library upf_export */
```

The following points explain the different sections of the exported Liberty power model:

- **Section1**: This section includes the basic design information in terms of the library and cell names for which the Liberty power model has been extracted and the software version used for the same.
- **Section2**: This section represents the library description. These are library level features and attributes.

The library level `voltage_map` attribute associates the voltage name with the relative voltage values. These specified voltage names are referenced by the `pg_pin` groups defined at the cellview level. The voltage map can be a combination of both the primary power and ground supplies and bias power and ground supplies. Voltage values in the Liberty power model should follow the voltage values of the top-level design power intent where model is integrated unless a corresponding error is flagged at the power intent verification stage by CLP.

- **Section3**: This section represents the cell group information and the related attributes. The cell group statement gives the name of the cell being described. It appears at the library group level. The Liberty power model exports the power intent of the design, therefore, the information in this group comprises of:
  - `is_macro_cell`: The attribute identifies whether a cell is a macro cell.
  - `switch_cell_type`: This attribute supports macro cells with internal switches present to generate internal power. The valid values for this attribute are `coarse_grain` and `fine_grain`.
  - Pin level attributes: The different pin level attributes, such as `pg_pin`, `pg_type`, `power_down_function`, `related_power_pin`, `related_ground_pin`, `input_signal_level` and `output_signal_level` describe the specific information extracted from the design schematic by Power Manager extractor for all the boundary ports (PG and data). The state of these attributes is dependent on the setup registration for PG nets, connectivity, and design topology.
  - Macro Cell Modeling: The attributes `is_isolated`, `isolation_enable_condition`, `switch_pin`, `switch_function`, `pg_function`, and `pg_types` are required for macro cell modeling. These are required to model the cells, which use low power special cells, such as isolation cells and power switch cells connected to boundary ports.

For more information on the 1801 commands, refer to *Liberty User Guides and Reference Manual Suite* (Version 2017.06).

7

Verifying Power Intent of a Design

After completing the power intent specification for your design, you need to verify the correctness of the power intent as per the design. When you verify the design by using Power Manager, the tool automatically runs CLP that reads the power intent and generates a report with appropriate messages.

The chapter includes the following sections:

- Preparing and Running CLP
- Checking Design Hierarchy
- Checking Power Intent
- Power Intent Verification Requirements

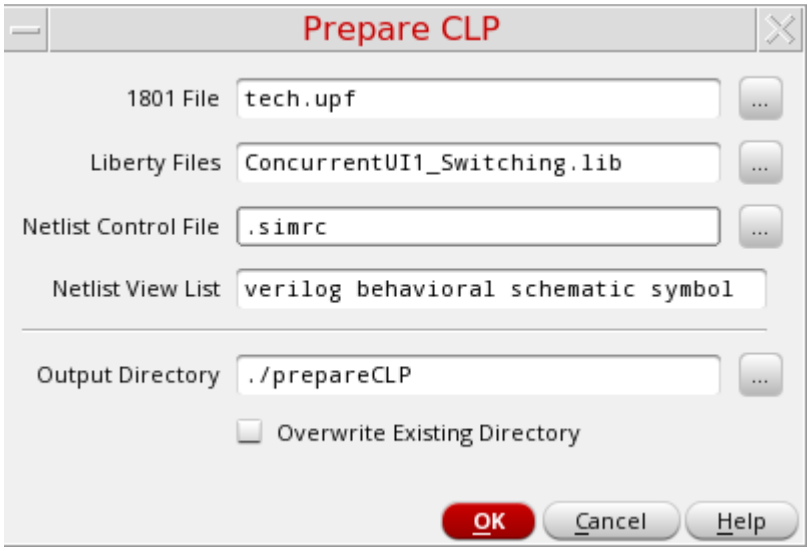
Preparing and Running CLP

Ensure that the following setup is done before you start verification using CLP:

- You have set the license for the Mixed-signal Option to Conformal Low Power (95127).
- You have set the path to the binary file of CLP, `lec`, in the UNIX path.
- You have checked the design hierarchy and ensured that there are no errors in the design. For more details about the recommended checks to be done, refer to Checking Power Intent.
- You have checked the power intent of the design to ensure its completeness. For more details about checking the completeness of power intent, refer to Checking Power Intent.
- You have defined the reference Verilog and Liberty files by including `read library` statements in the dofile (a file with set of commands required as inputs for power intent verification). This ensures that Conformal Low Power does not report the missing reference libraries. For more information on Conformal Low Power, refer to *Conformal Low Power User Guide*.

To verify the power intent, click *Power Manager – Run CLP* if you already have the required dofiles.

Otherwise, before running the CLP, use *Prepare CLP* to generate the 1801 file, Verilog netlist, and CLP dofile at the specified path.



- *1801 File* - This field is optional and only required if you want the tool to use a specific 1801 file for the CLP run. If the field is blank, you need to first extract the power intent to ensure that a new 1801 file is available to be exported and placed in the prepareCLP form.
- *Liberty Files* - This field is optional and only required if you want to include a Liberty file that is not a part of the setup. If the setup already has all the required Liberty files, these files are referred from the setup.
- *Netlist Control File* - Specify a control file for netlisting. This file contains flags used for netlist customization. The netlist customizations ensure that the power or ground information is available in the netlist in the desired format.

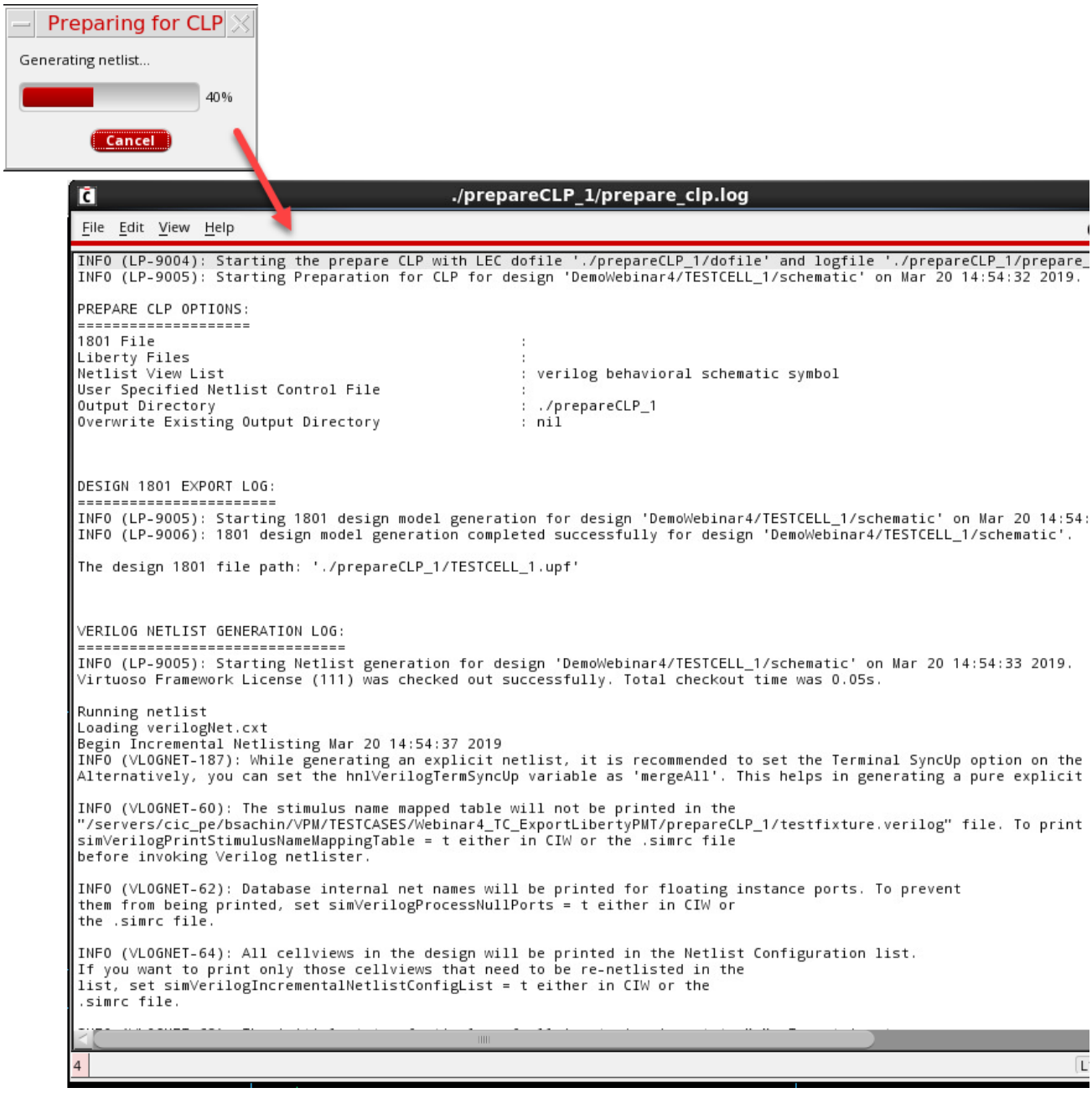
- *Netlist View List*- Specify a user switch view name list that is to be used for netlisting for the hierarchical designs.
- *Output Directory*- Specify the path to store a 1801 file, netlist, and dofile.
- *Overwrite Existing Directory*- Select this check box if you want to overwrite the existing output directory. If this option is deselected, the output directory is created with a unique name on each successive run.

Note: The *1801 File* and *Liberty Files* field do not require explicit user inputs, if a setup template file already exists and is loaded for the cellview to extract and verify the power intent.

The netlist generated by *Prepare CLP* is different from the general netlist created by NC-Verilog. This netlist includes a list of Verilog stub views for the macro models, single supply analog modules along with their power and ground pins, and the blocks that are binded to the existing power intent (1801). This makes the netlist more compatible with CLP. The log file generated, after the preparations for the CLP run are over, displays the results along with the following details:

- Date and time
- Design details
- Form field values
- Netlist/1801/dofile file paths
- Netlist error/warnings
- The ‘si.env’ file used for netlisting
- CLP log file path

- The CLP Run command for command line



Then, click *Run CLP* to start CLP and use the files created during CLP preparation, for power verification. The CLP run initiates when you click *OK*.

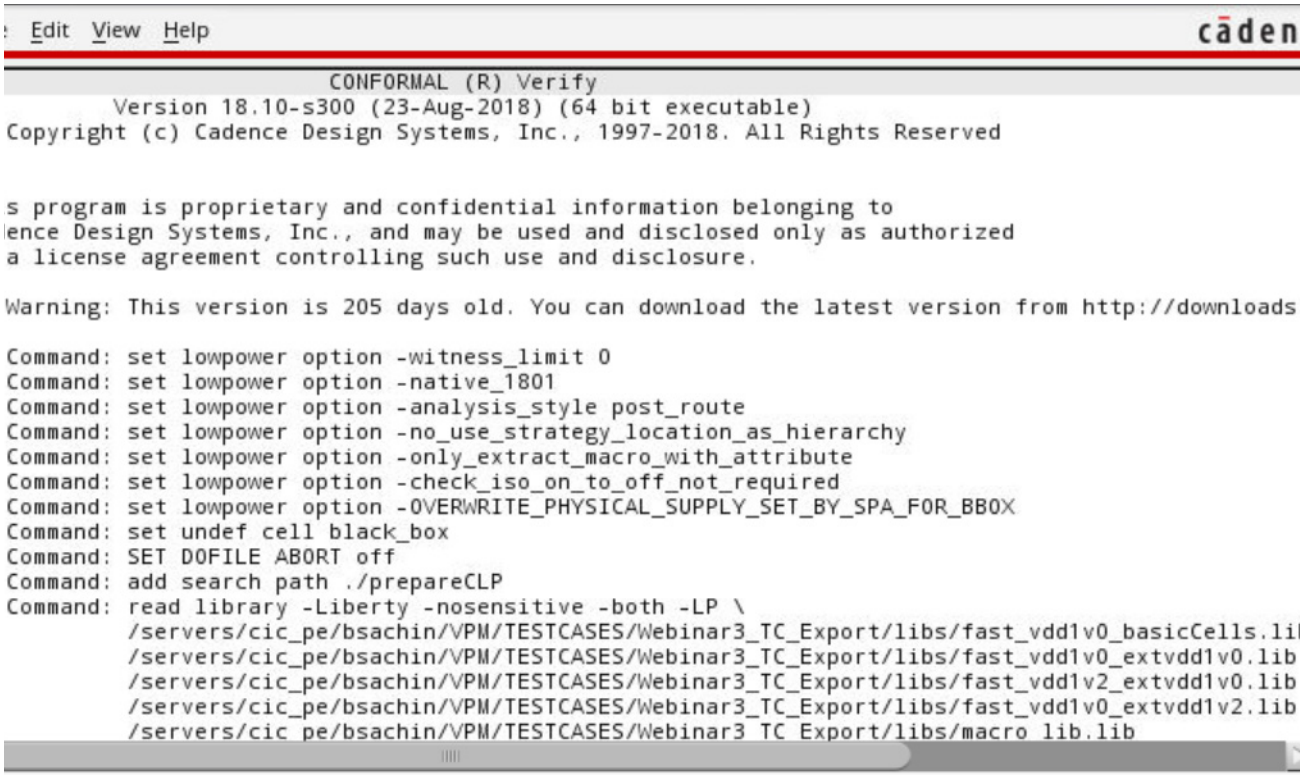
- *dofile Path* - Specify the path of the dofile.

- *Run Directory* - Specify the CLP run directory. The default value for this field is the current working directory.

This two-step process, which includes preparing for CLP and running CLP, lets you check the input files and resolve issues, if any, before proceeding for the CLP run. You can also run CLP from Power Manager by specifying a run directory and a CLP dofile. In addition, you can run CLP from the command line once the input files are generated using Power Manager.

- *Command* - `lec-nogui-lp-verify -pic -do dofile`

After the verification is complete, the generated report is displayed in a separate log window as shown below.



Checking Design Hierarchy

Before verifying the power intent for your design, it is recommended that you check the connectivity information in the design hierarchy by choosing *Check – Hierarchy*. The **Check Hierarchy** form is displayed. Select the appropriate check options on this form and click *OK*.

Important

For more details about the check options, refer to Check Hierarchy Form in the Virtuoso Schematic Editor user guide.This check is not run if you remove netSet properties from the design.

Checking Power Intent

Power intent can be defined as incomplete in any of the following scenarios:

- Incomplete power domains
  - Missing power net or ground net
  - Power or ground net is an internal net in a macro model
  - Missing base domain, when shut-off condition is present
  - Specified power or ground nets are not a part of design
- Incomplete low power rules
  - Incomplete isolation rules
  - Incomplete power switch rules
- Incomplete low power cells
  - Missing power or ground pins
  - Missing enable pins, specific for the isolation cells
- Missing off conditions
  - Missing off condition, when switchable domains are present
- Missing power modes
  - No power mode
  - Missing power mode in which all domains are on
  - Missing power mode in which software domains are off
- Unmapped ports of the design—All ports of the design must either belong to a power domain, floating ports, or feed-through port list, or the power intent is treated as incomplete

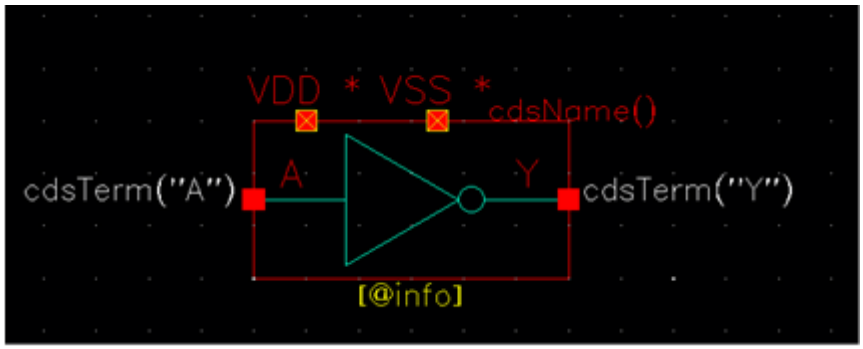
- Unmapped supply nets in the design—All supply nets must be a part of some power domain (either as direct Power/Ground net or as equivalent power/ground net), or the power intent is treated as incomplete

Power Intent Verification Requirements

The accurate power intent verification using CLP has the following mandatory requirements before proceeding for preparing and subsequently running CLP.

- `cmos_sch`, `schematic`, or `Verilog_PG` (Non-text) required as stop views for PDK cells (Standard/Special cells).
  - If `cmos_sch`, `schematic`, or `Verilog_PG` (symbol) exists with power and ground pins, they can be consumed as it is.

  A Verilog symbol view with the PG information is illustrated below.



- A netlist control file having flags for specific netlist customizations.

  If you explicitly specify a netlist control file during preparation for running CLP, set the following flags appropriately:
  - To ensure that the power and ground information is correctly captured in the netlist. This is required for correct power intent verification of the design.
    - `hnlInhConnUseDefSigName='t`
    - `vlogifDeclareGlobalNetLocal='t`
    - `hnlPrintInhConAtTop='t`
    - `hnlHonorInhConnEscapeName='t`
    - `hnlVerilogDeclareScalarSig='t`
    - `simVerilogEnableEscapeNameMapping= 't`

- `hnlMapNetInName`
- `hnlMapTermInName`

- To ensure that redundant information does not show at the module or instance level in the netlist. For example, information of primitive devices (mos/resistor/capacitors and diodes).

  `hnlUserIgnoreCVList`

- For creating the stub view (no module definition) for macro Liberty blocks, just an instance line with power/ground nets that is identified by traversing the schematic.

  `hnlUserStubCVList`

For more information about the flags used for netlist customization, refer to *Virtuoso NC Verilog Environment User Guide* and *Open Simulation System Reference*.

A

Environment Variables for Virtuoso Power Manager

This appendix describes public environment variables that you can use to customize the behavior of the Virtuoso Power Manager:

- addNotInGndPgFunction
- allowDomainWithoutElements
- allowEmptyDomains
- allowInoutLDOPins
- allowInoutMonitorPins
- allowInOutPortAsRegulated
- allowIsUnconnectedAsAttrInLiberty
- createExtractionLogFile
- createPinsOnImport
- delimiterForInternalPower
- extractionLogPath
- extractParasiticDiode
- lpDBGlobalSearchLibs
- lpDBGlobalSearchViews
- lpDBPurgeOnDesignPurge
- lsSingleRailInputPrefix
- powerDownFunctionForIOPorts
- printModelInSDA

- printSupplyNetInfo
- removeInternalNetFromPgFn
- separatorForBit
- specialCellInfoFile
- stopViewList
- switchPinForNoPortAtSwitchablePower
- switchViewList
- useANDForMultipleSupplies
- writeMixedFormatPST

addNotInGndPgFunction

```
lp addNotInGndPgFunction boolean { t | nil }
```

Description

Controls printing `pg_function` for ground switchable supplies. The `pg_function` value is printed as `VSS` if the variable is set to `nil`. The funtion is printed as `!VSS` if variable is set to `t`. The default value is `nil`.

GUI Equivalent

None

Examples

```
envSetVal("lp" "addNotInGndPgFunction" 'boolean nil)
```

allowDomainWithoutElements

```
lp allowDomainWithoutElements boolean { t | nil }
```

Description

Controls the creation of power domains exported in the 1801 file based on their constituent elements.

When allowDomainWithoutElements is `t`, only power domains without elements are exported in the 1801 file. If this variable is set to `nil`, the power domains that have instances/elements are exported in the 1801 file and the domains without any elements are not exported in the 1801 file.

The default value is `nil`.

GUI Equivalent

None

Examples

Default scenario

```
create_power_domain pd__vddB_init__vssB \
-elements { i_block }  \
-supply { primary ss__vddB_init__vssB } \
-supply { extra_supplies "" }

create_power_domain pd__vddA__vssA \
-include_scope  \
-supply { primary ss__vddA__vssA } \
-supply { extra_supplies "" }
```

Non Default scenario

```
create_power_domain pd__vddB_init__vssB \
-elements { i_block }  \
-supply { primary ss__vddB_init__vssB } \
-supply { extra_supplies "" }

create_power_domain pd__vddB__vssB \
```

```
-supply { primary ss__vddB__vssB } \
-supply { extra_supplies "" }

create_power_domain pd__vddA__vssB \
-supply { primary ss__vddA__vssB } \
-supply { extra_supplies "" }

create_power_domain pd__vddA__vssA \
-elements {.}  \
-supply { primary ss__vddA__vssA } \
-supply { extra_supplies "" }
```

allowEmptyDomains

```
lp allowEmptyDomains boolean { t | nil }
```

Description

Controls the printing of a supply or ground net as an internal power or internal ground pinif they are not associated with any power domain. The default value is `nil`. These supply or ground nets are not the related power or related ground pin of any signal pins.

GUI Equivalent

None

Examples

```
envSetVal("lp" "allowEmptyDomains" 'boolean nil)
```

allowInoutLDOPins

```
lp allowInoutLDOPins boolean { t | nil }
```

Description

Identifies supply nets, meeting the LDO net detection criteria, even when the nets are connected to `inout` pin/term, when the flag is set to `t`. The default value is `nil` and the extractor includes nets without a term or with an output term.

GUI Equivalent

None

Examples

```
envSetVal("lp" "allowInoutLDOPins" 'boolean nil)
```

allowInoutMonitorPins

```
lp allowInoutMonitorPins boolean { t | nil }
```

Description

Identifies supply nets, meeting the monitor net detection criteria, even when the nets are connected to `inout` pin/term, when the flag is set to `t`. The default value is `nil` and the extractor includes only the nets with the output term.

GUI Equivalent

None

Examples

```
envSetVal("lp" "allowInoutMonitorPins" 'boolean nil)
```

allowInOutPortAsRegulated

```
lp allowInOutPortAsRegulated boolean { t | nil }
```

Description

Considers supplies connected to the `inout` port as regulated. The default value is `nil`. By default, only supplies connected to output port or not connected to any port are considered as regulated supplies.

GUI Equivalent

None

Examples

```
envSetVal("lp" "allowInOutPortAsRegulated" 'boolean nil)
```

allowIsUnconnectedAsAttrInLiberty

```
lp allowIsUnconnectedAsAttrInLiberty boolean { t | nil }
```

Description

Prints the `is_unconnected` attribute as the main code in the Liberty file, when the flag is set to the default value `t`. When the environment variable is set to `nil`, the `is_unconnected` attribute is printed as comment in the Liberty file. When `allowIsUnconnectedAsAttributeInLiberty` is `true`, the Liberty file has the following user-defined attributes printed as a part of the Library group attributes of the exported Liberty file.

```
define ("is_unconnected", "pin", "boolean");

define ("is_unconnected", "pg_pin", "boolean");
```

GUI Equivalent

None

Examples

```
envSetVal("lp" "allowIsUnconnectedAsAttrInLiberty" 'boolean nil)
```

createExtractionLogFile

```
lp createExtractionLogFile boolean { t | nil }
```

Description

Prints extraction messages to a log file. The default value is `nil`.

GUI Equivalent

None

Examples

```
envSetVal("lp" "createExtractionLogFile" 'boolean nil)
```

createPinsOnImport

```
lp createPinsOnImport boolean { t | nil }
```

Description

Controls the creation of ports in the design while importing the 1801 file. This is based on the `Create_supply_port` command found in the input 1801 file.

The default value is `t`. When set to `t`, ports are created in the design. If set to `nil`, a supply net with a global net expression or supply local net is created by using the `Create_supply_port` and `Create_supply_net` commands in the input 1801 file based on the `resolveTopNets` argument of the vpmImportPowerIntent.

GUI Equivalent

None

Examples

```
envSetVal("lp" "createPinsOnImport" 'boolean nil)
```

delimiterForInternalPower

```
lp delimiterForInternalPower string { "___" }
```

Description

Specifies a delimiter for internal supplies that are referenced by the Liberty file generated by Power Manager. The default value is `"___"`.

GUI Equivalent

None

Examples

```
voltage_map( I0__gnd_int , 0.000000);
voltage_map( I0__vdd_int , 0.800000);

pg_pin(I0__vdd_int)  {
witch_function      : "!(Enable)";
voltage_name         : I0__vdd_int;
pg_function          : vdd;
pg_type              : internal_power;
direction            : internal;
}
pg_pin(I0__gnd_int)  {
switch_function      : "Sleep";
voltage_name         : I0__gnd_int;
pg_function          : gnd;
pg_type              : internal_ground;
direction            : internal;

pin(data)  {
related_ground_pin   : I0__gnd_int;
related_power_pin    : I0__vdd_int;
direction            : input;
```

extractionLogPath

```
lp extractionLogPath string { "." }
```

Description

Controls the printing of the extraction log file at the path provided. The default value is `"."` and the extraction log is generated in the current working directory.

GUI Equivalent

None

Examples

```
envSetVal("lp" "extractionLogPath" 'string ".abc/testlib/ext.log")
```

extractParasiticDiode

```
lp extractParasiticDiode boolean { t | nil }
```

Description

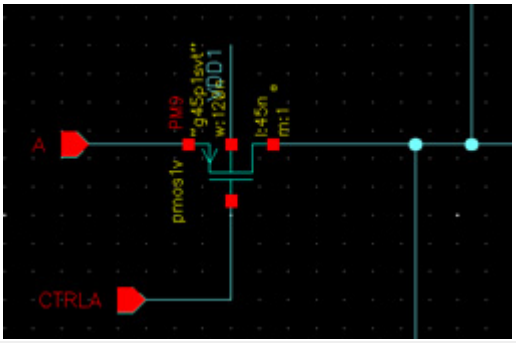
Controls the detection and printing of parasitic diode elements formed between the bulk terminal and source terminal, bulk terminal and drain terminal, or between MOS devices. When set to `t`, the parasitic diode information gets extracted in terms of the `antenna_diode_related_power_pins` and `antenna_diode_related_ground_pins` attributes corresponding to data pins at the source or drain end of a MOS device. The default value is `nil`.

GUI Equivalent

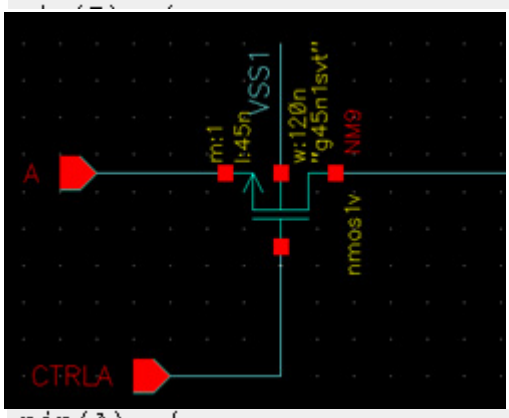
None

Examples

With `envSetVal("lp" "extractParasiticDiode" 'boolean t)`




```
pin(A)   {
         antenna_diode_related_power_pins : "VDD1";
         direction                : "input";
         related_power_pin        : "VDD1";
         related_ground_pin       : "VSS1";
}
```




```
pin(A)   {
         antenna_diode_related_ground_pins : "VSS1";
         direction                : "input";
         related_power_pin        : "VDD1";
         related_ground_pin       : "VSS1";
}
```

lpDBGlobalSearchLibs

```
lp lpDBGlobalSearchLibs string { "LibName" }
```

Description

Sets the library list look-up order for enabling the power view creation in a different library if the parent library is not editable. This enables the power view creation and subsequent access of the cellview.

By default, no library name is specified. The parent library is the location for power view creation.

Else, the specified library name or a list of library names are considered as the location for power view creation.

GUI Equivalent

None

Examples

In .cdsenv:

```
lp lpDBGlobalSearchLibs string { "DESIGN_LIB_LDO DESIGN_NEW_LIB_LDO" }
```

In .cdsinit or the CIW:

```
lp lpDBGlobalSearchLibs string { "DESIGN_LIB_LDO DESIGN_NEW_LIB_LDO" }
```

This would enable the creation and opening of the power view of the cell in the libraries `DESIGN_LIB_LDO` or `DESIGN_NEW_LIB_LDO`. The order of look-up to create and access the power view is from `DESIGN_LIB_LDO` until `DESIGN_NEW_LIB_LDO`. If the library `DESIGN_LIB_LDO` is not editable, the power view is created and accessed from library `DESIGN_NEW_LIB_LDO`.

lpDBGlobalSearchViews

```
lp lpDBGlobalSearchViews string { "power" }
```

Description

Sets the look-up order of the list of power view names for enabling the power view creation in a different library if the parent library is not editable. This enables the access of the power view created using `lpDBGlobalSearchLibs`, which is based on the power view name. The power view name is looked up in the same order for the power view creation in the library as defined in lpDBGlobalSearchLibs.

By default, `power` is specified as the power view name.

Else, the specified power view name or a list of power view names are considered as power views.

GUI Equivalent

None

Examples

In .cdsinit or the CIW:

```
lp lpDBGlobalSearchViews string { "pwr power" }
```

In .cdsenv:

```
lp lpDBGlobalSearchViews string { "pwr power" }
```

This would enable the access of the power view created using `lpDBGlobalSearchViews` in the specified library based on the view names `pwr` and `power`. The power view name `pwr` would be looked up first in the library. If it is not found, `power` would be searched for.

lpDBPurgeOnDesignPurge

```
lp lpDBPurgeOnDesignPurge boolean { t | nil }
```

Description

Controls the purging of the power view while closing the source design. The default value is `t`. By default, the power view is purged when the source design gets purged.

GUI Equivalent

None

Examples

```
envSetVal("lp" "lpDBPurgeOnDesignPurge" 'boolean nil)
```

lsSingleRailInputPrefix

```
lp lsSingleRailInputPrefix string { "int_" }
```

Description

For a single rail cell in case the modelling information is coming from 1801 special cell definition file, it would not contain information of the attribute `input_signal_level` corresponding to the liberty model. A virtual supply name, with the prefix `int_`, will be created from output supply of the single rail level shifter. The data ports of the single rail level shifter would thus be related to a supply set containing the virtual supply name and the name of the ground supply as related power and related ground.

GUI Equivalent

None

Examples

```
voltage_map( vdd_Int[0] , 0.800000);
         voltage_map( vdd_Int[1] , 0.800000);
         voltage_map( vdd_Int[2] , 0.800000);


pg_pin(vdd_Int[1])  {
voltage_name         : "vdd_Int[1]";
pg_type              : internal_power;
direction            : "internal";
switch_function      : "Enable[1]";
pg_function          : "vdd";
}

pin(in1)  {
direction            : "inout";
related_power_pin    : "vdd_Int[1]";
related_ground_pin   : "I0[int]gnd_int[1]";
                                      }
```

powerDownFunctionForIOPorts

```
lp powerDownFunctionForIOPorts boolean { t | nil }
```

Description

Specifies the `power_down_func` attribute for inout ports in a Liberty file. The default value is `nil`, in which case `power_down_func` is not printed for inout ports.

GUI Equivalent

None

Examples

```
envSetVal("lp" "powerDownFunctionForIOPorts" 'boolean t)
```

printBusBitDirectionInLiberty

```
lp printBusBitDirectionInLiberty boolean { t | nil }
```

Description

Controls the printing of the direction attribute for bus bits in the exported macro Liberty power model. The default value is `nil`, in which case the direction attribute is printed at the bus group level. When set to `t`, the direction attribute is printed for each individual bus bit.

GUI Equivalent

None

Examples

With the value of `printBusBitDirectionInLiberty` as `nil` ->

```
bus(OUT) {

bus_type              : CDS_BUS_1_To_0;
related_power_pin     : VDD;
related_ground_pin    : VSS;
power_down_function   : "!VDD + VSS";
}
pin(OUT[0])  {
related_power_pin     : VDD;
related_ground_pin    : VSS;
power_down_function   : "!VDD + VSS";
}

                                        }
```

With the value of `printBusBitDirectionInLiberty` as `t` ->

```
bus(OUT) {

bus_type              : CDS_BUS_1_To_0;
pin(OUT[1])  {
direction             : output;
```

```
related_power_pin    : VDD;
related_ground_pin   : VSS;
power_down_function  : "!VDD + VSS";
}
pin(OUT[0])  {
direction            : output;
related_power_pin    : VDD;
related_ground_pin   : VSS;
power_down_function  : "!VDD + VSS";
}

}
```

printModelInSDA

```
lp printModelInSDA boolean { t | nil }
```

Description

Adds model names to the exported UPF file when set to `t`. The default value is `nil`. By default, the command used is:

```
set_design_attributes -attribute top_ports_have_anon_supply 0
```

With value is set as `t`, the command used is:

```
set_design_attributes -models top -attribute top_ports_have_anon_supply 0
```

GUI Equivalent

None

Examples

```
envSetVal("lp" "printModelInSDA" 'boolean nil)
```

printSupplyNetInfo

```
lp printSupplyNetInfo boolean { t | nil }
```

Description

Prints the list of design nets that have been identified under any category of supply nets, such as `power`, `ground`, `exclude`, `monitor`, `pwell`, `nwell`, `deeppwell`, or `deepnwell` when the flag is set to `t`. The default value of the flag is `nil`.

GUI Equivalent

None

Examples

```
envSetVal("lp" "printSupplyNetInfo" 'boolean nil)
```

removeInternalNetFromPgFn

```
lp removeInternalNetFromPgFn boolean { t | nil }
```

Description

Removes internal supply or internal ground nets from `pg_function`. The default value of the flag is `t`. This behavior is not applied for the internal supply or internal ground nets that are related power or related ground of a signal pin. These internal supply or ground nets are still included in `pg_function`.

GUI Equivalent

None

Examples

When set to `nil`,

```
pg_pin(VDD_INT2)  {
            voltage_name         : "VDD_INT2";
            pg_type              : internal_power;
            direction            : "internal";
            pg_function          : "vdd_int";
        }
```

When set to `t`,

```
pg_pin(VDD_INT2)  {
            voltage_name         : "VDD_INT2";
            pg_type              : internal_power;
            direction            : "internal";
        }
```

separatorForBit

```
lp separatorForBit string { "<>" }
```

Description

Specifies separators for pin names of the bus bits. The default value is `"<>"`.

GUI Equivalent

None

Examples

```
envSetVal("lp" "separatorForBit" 'string "[]")
voltage_map( vdd_Int[0] , 0.800000);
         voltage_map( vdd_Int[1] , 0.800000);
         voltage_map( vdd_Int[2] , 0.800000);


pg_pin(vdd_Int[1])  {
voltage_name          : "vdd_Int[1]";
pg_type               : internal_power;
direction             : "internal";
switch_function       : "Enable[1]";
pg_function           : "vdd";
}

pin(in1)  {
direction             : "inout";
related_power_pin     : "vdd_Int[1]";
related_ground_pin    : "I0[int]gnd_int[1]";
                                      }
```

specialCellInfoFile

```
lp specialCellInfoFile string ""
```

Description

Provides a mechanism to parse the 1801 special cell definition file to the 1801 extractor externally, which is outside the Power Manager setup. The 1801 extractor reads the 1801 special cell definition file accordingly and reads the modelling information from the same.

GUI Equivalent

None

Examples

```
envSetVal("lp" "specialCellInfoFile" 'string "./spcells.upf")
```

stopViewList

```
lp stopViewList string { "symbol symbol_xform auCdl auLvs hspiceD spectre" }
```

Description

Specifies a space-separated list of cellview names that is used while elaborating the design under test to identify the view at which the traversal has to be stopped. While traversing a design hierarchy, the tool stops moving further when it finds any one of the cellviews given in this list. Power Manager uses the stop view list for design traversal during the power intent extraction phase and also to create the required design objects during the 1801 import flow.

The default value is `"symbol symbol_xform auCdl auLvs hspiceD spectre"`.

GUI Equivalent

None

Examples

In .cdsenv:

```
lp stopViewList string { "symbol symbol_xform aucdl" }
```

In .cdsinit or the CIW:

```
envSetVal("lp" "stopViewList" 'string "symbol symbol_xform aucdl")
```

switchPinForNoPortAtSwitchablePower

```
lp switchPinForNoPortAtSwitchablePower boolean { t | nil }
```

Description

Prints the `switchPin` attribute for enable pin even if there is no port for power switch output, when set to `t`, which is the default value.

GUI Equivalent

None

Examples

If value is `nil`:

```
pin(Enable)  {
direction            : input;
related_power_pin    : vdd;
related_ground_pin   : I0__net7;
}
```

If value is `t`:

```
pin(Enable)  {
direction            : input;
switch_pin           : true;
related_power_pin    : vdd;
related_ground_pin   : I0__net7;
}
```

switchViewList

```
lp switchViewList string { "viewnames" }
```

Description

Specifies a space-separated list of cellview names that is used while reviewing the design under test to control the order in which the design hierarchy is traversed. Power Manager uses the switch view list for design traversal during the power intent extraction phase and also to create the required design objects during the 1801 import flow.

The default value is `""`.

GUI Equivalent

None

Examples

In .cdsenv:

```
lp switchViewList string { "cmos_sch schematic symbol" }
```

In .cdsinit or the CIW:

```
envSetVal("lp" "switchViewList" 'string "cmos_sch cmos.sch schematic symbol")
```

useANDForMultipleSupplies

```
lp useANDForMultipleSupplies boolean { t | nil }
```

Description

Prints the multiple supplies in the output Liberty file as `ExtVDD&VDD`, when set to `t`. When the cdsenv is set to `nil`, multiple supplies are printed as `ExtVDD+VDD`.

GUI Equivalent

None

Examples

```
envSetVal("lp" "useANDForMultipleSupplies" 'boolean nil)
```

writeMixedFormatPST

```
lp writeMixedFormatPST boolean { t | nil }
```

Description

Enables the use of syntax or semantics for writing mixed power states in the exported 1801power model file. This allows the printing of a mix of Unified Power Format 2.0 and Unified Power Format 1.5 for specifying the power states in the extracted 1801 power model. The default value is `nil`. If the environment variable is set to `nil`, the power states are written using the `add_power_state` command only. The multiple supplies are printed as `ExtVDD+VDD`. When set to `t`, power states are written by using the following commands:

- `add_power_state`
- `create_pst`
- `add_pst_state`

GUI Equivalent

None

Examples

When Set to `t`

```
##### Power States ##################################################
add_power_state ss__vddA__vssA -supply -state { on_1p0 -supply_expr { (power == {
FULL_ON 1.000000 }) && (ground == { OFF }]}}
add_power_state ss__vddA__vssB -supply -state { on_1p0 -supply_expr { (power == {
FULL_ON 1.000000 }) && (ground == { OFF }]}}
add_power_state ss__vddB__vssB -supply -state { on_1p5 -supply_expr { (power == {
FULL_ON 1.500000 }) && (ground == { OFF }]}}
add_power_state ss__vddB_init__vssB -supply -state { on_1p5 -supply_expr { (power
== { FULL_ON 1.500000 }) && (ground == { OFF }]}} -state { off_0p0 \
-supply_expr { (power == { OFF }) && (ground == { OFF })}}
add_power_state pd__vddA__vssA -domain -state { mode0 -logic_expr {ss__vddB__vssB
== on_1p5 && ss__vddA__vssB == on_1p0 && ss__vddA__vssA == on_1p0 && \
ss__vddB_init__vssB == on_1p5 }} -state { mode1 -logic_expr { ss__vddB__vssB ==
on_1p5 && ss__vddA__vssB == on_1p0 && ss__vddA__vssA == on_1p0 && \
ss__vddB_init__vssB == off_0p0}}
```

When Set to `nil`

```
##### Power States ##################################################
add_power_state ss__vddA__vssA -state { on_1p0 -supply_expr { power == { FULL_ON
1.000000 }}} -state { on_0p0 -supply_expr { ground == { FULL_ON 0.000000 }}}
add_power_state ss__vddA__vssB -state { on_1p0 -supply_expr { power == { FULL_ON
1.000000 }}} -state { on_0p0 -supply_expr { ground == { FULL_ON 0.000000 }}}
add_power_state ss__vddB__vssB -state { on_1p5 -supply_expr { power == { FULL_ON
1.500000 }}} -state { on_0p0 -supply_expr { ground == { FULL_ON 0.000000 }}}
add_power_state ss__vddB_init__vssB -state { on_1p5 -supply_expr { power == {
FULL_ON 1.500000 }}} -state { on_0p0 -supply_expr { ground == { FULL_ON 0.000000
}}} \
-state { off_0p0 -supply_expr { power == { OFF }}}
##### PST ###########################################################
create_pst cellA -supplies { ss__vddB__vssB.power ss__vddB__vssB.ground
ss__vddA__vssB.power ss__vddA__vssA.ground ss__vddB_init__vssB.power}
add_pst_state mode0 -pst cellA -state {on_1p5 on_0p0 on_1p0 on_0p0 on_1p5 }
add_pst_state mode1 -pst cellA -state {on_1p5 on_0p0 on_1p0 on_0p0 off_0p0 }
```

B

Virtuoso Power Manager SKILL Functions

This chapter describes the public SKILL functions in Power Manager.

vpmDefinePowerSwitchInstance

```
vpmDefinePowerSwitchInstance(
     t_cellName
     t_instName
     g_type
     g_stage1Enable
     [ ?stage2Enable g_stage2Enable ]
     [ ?stage1Output g_stage1Output ]
     [ ?stage2Output g_stage2Output ]
     [ ?power g_power ]
     [ ?powerSwitchable g_powerSwitchable ]
     [ ?ground g_ground ]
     [ ?groundSwitchable g_groundSwitchable ]
     )
     => t / nil
```

Description

Registers transistor instances as power or ground switches.

Arguments

t_cellName	The name of the parent cell name of power switch instance.
t_instName	The name of the power switch instance.
g_type	Specifies `header` for power switch or `footer` for ground switch.
g_stage1Enable	The stage 1 enable expression for power switch instance.
g_stage2Enable	The stage 2 enable expression for power switch instance.
g_stage1Output	The stage 1 output expression for power switch instance.
g_stage2Output	The stage 2 output expression for power switch instance.
g_power	The power pin of power switch instance.
g_powerSwitchable	The switchable pin of power switch instance.
g_ground	The ground pin of ground switch instance.
g_groundSwitchable	The switchable pin of ground switch instance.

Value Returned

`t`	The instances are registered successfully.
`nil`	The instances could not be registered.

Example

```
vpmDefinePowerSwitchInstance("analog_blk" "M0" ?type "header" ?power "S"
?powerSwitchable "D" ?stage1Enable "!G")
```

- `M0` is power switch instance in the `analog_blk` cell.

vpmExportDotLib

```
vpmExportDotLib(
     x_session_id
     t_file_dot_lib
     )
     => t / nil
```

Description

Exports the extracted power intent in the `.lib` file format.

Arguments

x_session_id	The ID of the window for which you want to export the power intent.
t_file_dot_lib	The name of the file for exporting the power intent.

Value Returned

`t`	The file is exported successfully.
`nil`	The file could not be exported.

Example

```
vpmImportPowerIntentSetup( "test_pg" "pass_gate_SC" "schematic" "lpSetup.il")
session = (vpmExtractPowerIntent  "test_pg" "pass_gate_SC" "schematic")
vpmExportDotLib (session "libModel.lib")
```

Exports the power intent as the Liberty Power Model `libModel.lib` file.

vpmExportPowerIntent

```
vpmExportPowerIntent(
     x_session_id
     t_file_1801
     )
     => t / nil
```

Description

Exports the 1801 Design Model for a design. Ensure that the design has been extracted by using vpmExtractPowerIntent or the GUI options in Virtuoso Schematic Editor. This function requires a session ID, which is generated upon successful power intent extraction for the design.

Arguments

x_session_id	The ID of the window for which you want to export the power intent.
t_file_1801	The name of the file for exporting the power intent.

Value Returned

`t`	Returns `t` if the power intent is successfully exported.
`nil`	Returns `nil` otherwise.

Example

```
vpmImportPowerIntentSetup( "test_pg" "pass_gate_SC" "schematic" "lpSetup.il")
session = (vpmExtractPowerIntent "test_pg" "pass_gate_SC" "schematic")
vpmExportPowerIntent (session "designmodel.upf")
```

vpmExportPowerModel

```
vpmExportPowerModel(
     x_sessionID
     t_output_file_name
     )
     => t
```

Description

Exports the 1801 power model for a design. Ensure that the design has been extracted by using `vpmExtractPowerIntent`, or using the GUI options in Virtuoso Schematic Editor. This function requires a session ID, which is generated upon successful power intent extraction for the design.

Arguments

x_sessionID	The return value of `vpmExtractPowerIntent`.
t_output_file_name	The name of the file for exporting the 1801 power model.

Value Returned

`t`	Returns `t` if the 1801 power intent model is successfully exported to the file. Displays an error message.

Example

```
vpmImportPowerIntentSetup( "test_pg" "pass_gate_SC" "schematic" "lpSetup.il")
session = (vpmExtractPowerIntent "test_pg" "pass_gate_SC" "schematic")
vpmExportPowerModel (session "designmodel.upf")
```

vpmExportPowerIntentSetup

```
vpmExportPowerIntentSetup(
     t_setup_file_name
     [ ?overwriteExisting g_overwriteExisting ]
     [ ?filter g_filter ]
     [ ?lpDBLibName g_lpDBLibName ]
     [ ?lpDBCellName g_lpDBCellName ]
     [ ?lpDBViewName g_lpDBViewName ]
     )
     => t / nil
```

Description

Exports the loaded setup to a specified file. This file can be verified for the setup options that the Power Manager has used during import/export of power intent.

Arguments

t_setup_file_name	Name of the file to which the loaded setup is exported.
g_overwriteExisting	Specifies if the exported file can overwrite an existing file with the same name, if any.
g_filter	Specifies which setup options out of `environmentSetupOptions`, `userSetupOptions`, or `allSetupOptions` are to be exported to the filename specified in `t_output_file_name`.
g_lpDBLibName	Name of the library in which the power view is generated.
g_lpDBCellName	Name of the cell in which the power view is generated.
g_lpDBViewName	Name of the power view generated.

Value Returned

`t`	Returns `t` if the loaded power intent setup is successfully exported to the file specified.
`nil`	Returns `nil` otherwise.

Example

```
vpmExportPowerIntentSetup( "test_pg" "pass_gate_SC" "schematic" "lpSetup.il"
?overwriteExisting t ?filter "allSetupOptions")
```

vpmExtractPowerIntent

```
vpmExtractPowerIntent(
     t_lib
     t_cellName
     t_viewName
     )
     => session_id / nil
```

Description

Extracts the power intent from the given design. It also identifies sub-hierarchical blocks in the top-level design, which need macro model extraction or are associated with a 1801 file.

Argument

t_lib	Name of the library of the cellview to be extracted.
t_cellName	Name of the cell of the cellview to be extracted.
t_viewName	View name from which power intent is to be extracted.

Value Returned

session_id	Returns the session id if the power intent model is successfully extracted.
nil	Returns nil otherwise.

Example

```
vpmExtractPowerIntent ("test_pg" "pass_gate_SC" "schematic")
```

vpmGenerateSigInfo

```
vpmGenerateSigInfo(
     t_libName
     t_cellName
     t_viewName
     t_sigInfoFilePath
     )
     => session_id / nil
```

Description

Adds the signal type and voltage values for all the canonical nets in the design. If a design net spans across multiple levels in the hierarchy, the net at the highest level of the hierarchy is considered as canonical.

Argument

t_libName	Name of the design cellview.
t_cellName	
t_viewName	
t_sigInfoFilePath	Path of the text file that is created by the SKILL function that contains the details of the canonical design nets. For each canonical net listed in the file, its details include the signal type and net voltages. If no file path is specified, the function creates the file by the name of *t_libName_t_cellName.siginfo.txt* in the current working directory.

Value Returned

`t`	Returns `t` if it is able to successfully generate the information for canonical nets.
`nil`	Returns `nil` in one of these situations: ■ The `t_libName`, `t_cellName`, `t_viewName` are not strings or are empty strings. ■ The `t_libName`, `t_cellName`, `t_viewName` do not point to a valid and readable Open Access cellview. ■ One of the `t_libName`, `t_cellName`, or `t_viewName` argument is `nil`. ■ The `t_sigInfoFilePath` references to a directory location that is not writable. ■ Appropriate licenses required by the function are not available. ■ Power Manager Setup is not loaded for the specified design.

Example

```
when(ret
   ;; If setup is loaded successfully, try to generate sigInfo
   ret = vpmGenerateSigInfo( "testlib" "adcTop" "schematic" )
   when(ret
      ;; If the API run was successful it should have created a file by the name
of testlib_adcTop.siginfo.txt;
      ;; Open it in a view window to view the results
      view( "./testlib_adcTop.siginfo.txt")
   )
)
```

vpmImportPowerIntent

```
vpmImportPowerIntent(
     t_libname
     t_cellname
     t_viewname
     t_viewlist
     t_file1801Path
     [ ? resolveTopNets g_resolveTopNets ]
     )
     => t / nil
```

Description

Imports the power intent for any cellview. Apart from this, in the design hierarchy, the SKILL function creates the netSet properties and resolves tie connections. It also creates supply pins corresponding to the power domain nets and global nets.

Note: This function verifies if the Virtuoso Schematic Editor XL license is checked out. Otherwise, it checks out the license and releases license before exiting.

Alternatively, you can import the power intent by using *Power Manager – Import Power Intent*. For details, refer to Importing the Power Intent of a Design.

Arguments

t_libname	Name of the library of the cellview to be imported or exported for power intent.
t_cellname	Name of the cell to be imported or exported for power intent.
t_viewname	Name of the view to be imported or exported for power intent.
t_viewlist	Switch view list specifying the order of traversal of hierarchical schematic to create the required OA objects.
t_file1801Path	Name or path of the 1801 file to be imported.
g_resolveTopNets	Boolean option controlling the generation of supply nets in design as the global net expression when set to `nil` or a local net when set to `t`.

Value Returned

`t`	Returns `t` if the power intent setup is successfully imported.
`nil`	Returns `nil` otherwise.

Example

```
vpmImportPowerIntent( "test_pg" "pass_gate_SC" "schematic" "schematic symbol"
"import.upf" )
```

vpmImportPowerIntentSetup

```
vpmImportPowerIntentSetup(
     t_libname
     t_cellname
     t_viewname
     t_setup_file_name
     [ ? lpDBLibName g_lpDBLibName ]
     [ ? lpDBCellName g_lpDBCellName ]
     [ ? lpDBViewName g_lpDBViewName ]
     )
     => t / nil
```

Description

Imports a SKILL file that contains the user-defined setup. This has the attributes required to extract the desired information from the design cellview for importing or exporting the power intent.

Arguments

t_libname	Name of the library of the cellview to be imported or exported for power intent.
t_cellname	Name of the cell to be imported or exported for power intent.
t_viewname	Name of the view to be imported or exported for power intent.
t_setup_file_name	Name of the user-defined setup file.
g_lpDBLibName	Name of the library of the cellview in which the power view is generated.
g_lpDBCellName	Name of the cell in which the power view is generated.
g_lpDBViewName	Name of the power view in the cellview.

Value Returned

`t`	Returns `t` if the power intent setup is successfully imported or loaded.
`nil`	Returns `nil` otherwise.

Example

```
vpmImportPowerIntentSetup( "test_pg" "pass_gate_SC" "schematic" "lpSetup.il")
```

vpmLoadInDesignViolations

```
vpmLoadInDesignViolations(
     t_libname
     t_cellname
     t_viewname
     t_vdbFilePath
     )
     => t / nil
```

Description

Loads the specified violation database file that contains the violation details for a previous check on a design. As the SKILL function processes each violation from the violations file, it creates markers on relevant design objects. For details, refer to Loading the Violations Database.

Arguments

t_libname	Name of the library of the cellview to be imported or exported for power intent.
t_cellname	Name of the cell to be imported or exported for power intent.
t_viewname	Name of the view to be imported or exported for power intent.
t_vdbFilePath	Name or path of the 1801 file to be imported.

Value Returned

t	Returns t if the violations are loaded successfully.
nil	Returns nil in one of these situations: ■ t_libName, t_cellName, t_viewName are not strings, or are empty strings. ■ t_libName, t_cellName, t_viewName do not point to a valid and readable Open Access cellview. ■ One of the t_libName, t_cellName, t_viewName, argument is nil, the user gets a SKILL error. ■ The t_libName, t_cellName, and t_viewName do not match the library, cell, and view name of the design for which the violation file was generated. ■ The t_vdbFilePath references are non-existent or non-readable file. ■ Appropriate licenses required by the SKILL function are not available.

Example

```
ret = vpmLoadInDesignViolations( "testlib" "adcTop" "schematic"  "./
testlib.adcTop.vdb")
```

vpmRemoveImportedPowerIntent

```
vpmRemoveImportedPowerIntent(
     t_libName
     t_cellName
     t_viewName
     )
     => t_viewlist/ nil
```

Description

Removes the entire power intent information that was imported for a cellview. This function also removes the netSet properties, tie connections, supply pins, and so on.

For more details, refer to Importing the Power Intent of a Design.

This function is used only when the Schematic XL license is checked out. Therefore, it first verifies whether the license is already checked out and if it is not, it tries to check out the license. The function also releases the license before exiting.

The view list specified by `t_viewlist` is used for processing the removal of imported objects for the view type instantiated in the top cellview.

Argument

t_libname	Name of the library of the cellview for which the power intent will be removed
t_cellname	Name of the cell of the cellview for which the power intent will be removed
t_viewname	Name of the view of the cellview for which the power intent will be removed
t_viewlist	List of the view names to be processed for removing the power intent information.

Value Returned

t_viewlist	Returns the list of the view names to be processed for removing the power intent information.

`nil`	Returns `nil` otherwise.

Examples

```
vpmRemoveImportedPowerIntent( "test_pg" "pass_gate_SC" "schematic" "schematic"
"symbol")
```

vpmRemovePowerSwitchInstance

```
vpmRemovePowerSwitchInstance(
     t_cellName
     t_instName
     )
     => t / nil
```

Description

Removes registered transistor instances as power or ground switches.

Arguments

t_cellName	The name of the parent cell name of power switch instance.
t_instName	The name of the power switch instance.

Value Returned

t	The instances are removed successfully.
nil	The instances could not be removed.

Example

```
vpmRemovePowerSwitchInstance("analog_blk" "M0" )
```

M0 is an instance in the analog_blk cell, which had been registered as a power switch using SKIL API vpmDefinePowerSwitchInstance. This instance will not be treated as a power switch any more.

vpmRunInDesignChecks

```
vpmRunInDesignChecks(
     t_libName
     t_cellName
     t_viewName
     [ ?lprcFilePath t_lprcFilePath ]
     [ ?vdbFilePath g_vdbFilePath ]
     )
     => t / nil
```

Description

Runs the In-Design Checks on a design specified by library, cell, and view.

Arguments

t_libname	Name of the design cellview.
t_cellname	
t_viewname	
t_lprcFilePath	Path of the In-Design Checks log file that reports the summary and details of violations. This is an optional argument. If a value is not specified, the function creates the file by the name of *t_libName_t_cellName.inDesign.log* in the current working directory.
g_vdbFilePath	Name of the violation database (.vdb). It contains the violation details for the current run. It is an optional argument. If none is specified, the function creates the file by the name of *t_libName_t_cellName.vdb* in the current working directory.

Value Returned

t	Returns t if Power Manager has been run successfully.
nil	Returns nil in one of these situations: ■ The t_libName, t_cellName and t_viewName are not strings or are empty strings. ■ The t_libName, t_cellName, t_viewName do not point to a valid and readable Open Access cellview. ■ One of the t_libName, t_cellName, or t_viewName argument is nil. ■ If t_lprcFilePath references a directory location that is not writable. ■ If t_vdbFilePath references to a directory location that is not writable. ■ Appropriate licenses required by the function are not available. ■ The setup information is not loaded for the specified design.

Example

```
when(ret
   ;; If setup is loaded successfully, try to run checks
   ret = vpmRunInDesignChecks( "testlib" "adcTop" "schematic" )
   when(ret
      ;; If checks are successful it should have created a file by the name of
testlib_adcTop.inDesign.log;
      ;; Open it in a view window to view the results
      view( "./testlib_adcTop.inDesign.log")
   )
)
```

vpmSetInDesignBrowserOptions

```
vpmSetInDesignBrowserOptions(
     w_window
     t_openMode
     t_openLocation
     x_hierarchyDepth
     [ ?hierarchicalScope b_hierarchicalScope ]
     => t / nil
```

Description

Sets the Annotation Browser options for viewing the design violations.

Arguments

w_window	A valid GE window where the Power Manager application is running. When a valid window ID is a specified, the function sets the specified options only for the specific Annotation Browser assistant occurrence.
t_openMode	The mode in which the marker cellview is opened when you choose to navigate the marker using the *Open* option in the Annotation Browser. The valid values are `edit` and `read`.
t_openLocation	The location where the marker cellview is opened when you choose to navigate the marker using the *Open* option in the Annotation Browser. The valid values are `current tab`, `new tab`, or `new window`.
x_hierarchyDepth	The valid values can be an integer between `0` and `32`.
b_hierarchicalScope	The valid values are `t` and `nil`. It is an optional argument. If set to `t`, it sets the Annotation Browser scope value to `Curren Cellview To Depth` and the In-Design Checks markers in the current cellview hierarchy are displayed. When set to `nil`, then it sets the Annotation Browser scope value to `Current Cellview Only` and only the In-Design Checks markers in the current cellview are displayed.

Value Returned

`t`	Returns `t` if the Annotation Browser options have been set successfully.
`nil`	Returns `nil` if the Annotation Browser options could not be set successfully.

Example

```
when(window = hiGetCurrentWindow()
  when(saSetBroserOptions(?window window ?hierarchicalScope t ?hierarchyDepth 32)
     println("Successfully set browser options")
  )
)
```

C

1801 Support in Power Manager

General 1801 Command Support

In the IC6.1.8 and ICADVM20.1 releases, Power Manager supports 1801 commands as explained below:

- All 1801 commands are imported from the given 1801 file. The schematic connectivity in the 1801 Import flow happens according to the commands supported by the Power Manager.
- All supported commands and arguments are read from the tool and exported to a 1801 file.

The following table lists all the general 1801 commands and explains how they are supported in the 1801 import and export processes of Power Manager. All supported commands are indicated with an `S` and the unsupported commands with a `U`.

Command	Options	Valid in IEEE 1801 Version		Comments	Support for 1801 Import	Support for 1801 Export
create_power_domain		2.0	2.1		S	S
	-elements	2.0	2.1		S	S
	-include_scope	2.0	2.1	Deprecated in 2.1	S	S
	-supply	2.0	2.1		S	S
	-scope	2.0	2.1	Deprecated in 2.1	S	S
	-update	2.0	2.1		S	S
create_supply_port		2.0	2.1		S	S
	-direction	2.0	2.1		S	S
create_supply_net		2.0	2.1		S	S
	-resolve	2.0	2.1		S	S
connect_supply_net		2.0	2.1		S	S
	-ports	2.0	2.1		S	S
create_supply_set		2.0	2.1		S	S
	-function	2.0	2.1		S	S

Command	Options	Valid in IEEE 1801 Version		Comments	Support for 1801 Import	Support for 1801 Export
	-update	2.0	2.1		S	S
associate_supply_set		2.0	2.1		S	S
	-handle	2.0	2.1		S	S
set_port_attributes		2.0	2.1		S	S
	-applies_to	2.0	2.1		S	S
	-ports	2.0	2.1		S	S
	-elements	2.0	2.1		S	S
	-attribute	2.0	2.1		S	S
	-driver_supply	2.0	2.1		S	S
	-receiver_supply	2.0	2.1		S	S
	-repeater_supply	2.0	2.1	Deprecated in 2.1	S	S
add_port_state		2.0	2.1		U	S
	-state	2.0	2.1	Not needed for 1801 Import	U	S
add_pst_state		2.0	2.1		U	S
	-pst	2.0	2.1	Not needed for 1801 Import	U	S
	-state	2.0	2.1	Not needed for 1801 Import	U	S
create_pst		2.0	2.1		U	S
	-supplies	2.0	2.1	Not needed for 1801 Import	U	S
create_power_switch		2.0	2.1		S	S
	-output_supply_port	2.0	2.1		S	S

Command	Options	Valid in IEEE 1801 Version		Comments	Support for 1801 Import	Support for 1801 Export
	-input_supply_port	2.0	2.1		S	S
	-domain	2.0	2.1		S	S
	-control_port	2.0	2.1		S	S
	-on_state	2.0	2.1		S	S
	-off_state	2.0	2.1		S	S
	-supply_set	2.0	2.1		S	S
set_level_shifter		2.0	2.1		S	S
	-domain	2.0	2.1		S	S
	-elements	2.0	2.1		S	S
	-source	2.0	2.1		S	S
	-sink	2.0	2.1		S	S
	-applies_to	2.0	2.1		S	S
	-no_shift	2.0	2.1		S	S
	-location	2.0	2.1		S	S
	-update	2.0	2.1		S	S
	-input_supply_set	2.0	2.1		S	S
	-output_supply_set	2.0	2.1		S	S
	-internal_supply_set	2.0	2.1		S	S

Command	Options	Valid in IEEE 1801 Version		Comments	Support for 1801 Import	Support for 1801 Export
set_isolation		2.0	2.1		S	S
	-domain	2.0	2.1		S	S
	-elements	2.0	2.1		U	S
	-source	2.0	2.1		U	S
	-sink	2.0	2.1		U	S
	-diff_supply_only	2.0	2.1		S	S
	-isolation_supply_net	2.0	2.1		S	S
	-update	2.0	2.1		S	S
	-applies_to	2.0	2.1		S	S
	-clamp_value	2.0	2.1		S	S
	-no_isolation	2.0	2.1		S	S
	-isolation_signal	2.0	2.1		S	S
	-isolation_sense	2.0	2.1		S	S
	-location	2.0	2.1		S	S
load_upf		2.0	2.1		S	S
	-scope	2.0	2.1		S	S
set_scope		2.0	2.1		S	S
	Instance	2.0	2.1		S	S
begin_power_model			2.1		U	S
	power_model_name		2.1		U	S
	-for model_list		2.1		U	S
add_power_state			2.1		U	S
	-state		2.1		U	S
	-supply_expr		2.1		U	S
	-logic_expr		2.1		U	S
	-simstate		2.1		U	S

Command	Options	Valid in IEEE 1801 Version		Comments	Support for 1801 Import	Support for 1801 Export
end_power_model			2.1		U	S

1801 Special Cell Definition Command Support

The following table lists all the 1801 special cell definition commands and explains how they are supported in the 1801 import and export processes of Power Manager. All supported commands are indicated with an `S` and the unsupported commands with a `U`.

Command	Options	Valid in IEEE 1801 Version		Comments	Support for 1801 Import	Support for 1801 Export
define_isolation_cell		2.0	2.1		S	S
	-cells	2.0	2.1		S	S
	-power	2.0	2.1		S	S
	-ground	2.0	2.1		S	S
	-enable	2.0	2.1		S	S
	-power_switchable	2.0	2.1		S	S
	-ground_switchable	2.0	2.1		S	S

Command	Options	Valid in IEEE 1801 Version		Comments	Support for 1801 Import	Support for 1801 Export
define_level_shifter_cell		2.0	2.1		S	S
	input_voltage_range	2.0	2.1		S	S
	-output_voltage_range	2.0	2.1		S	S
	-direction	2.0	2.1		S	S
	-input_power_pin	2.0	2.1		S	S
	-output_power_pin	2.0	2.1		S	S
	-input_ground_pin	2.0	2.1		S	S
	-output_ground_pin	2.0	2.1		S	S
	-ground	2.0	2.1		S	S
	-power	2.0	2.1		S	S
	-enable	2.0	2.1		S	S
define_power_switch_cell		2.0	2.1		S	S
	-cells	2.0	2.1		S	S
	-type	2.0	2.1		S	S
	-stage_1_enable	2.0	2.1		S	S
	-stage_1_output	2.0	2.1		S	S
	-power_switchable	2.0	2.1		S	S
	-power	2.0	2.1		S	S
	-ground_switchable	2.0	2.1		S	S
	-ground	2.0	2.1		S	S

Liberty Attributes Support

The following table lists all the supported Liberty attributes and explains how they are supported in the 1801 export processes of the Power Manager. All supported commands are indicated with an `S` and the unsupported commands with a `U`.

Attribute	Valid in Liberty Versions		Comments	Support for Export Liberty Power Model
is_macro_cell	2014.09	2017.06		S
voltage_map	2014.09	2017.06		S
pg_pin	2014.09	2017.06		S
pg_type	2014.09	2017.06		S
direction	2014.09	2017.06		S
switch_function	2014.09	2017.06		S
related_power	2014.09	2017.06		S
is_isolated	2014.09	2017.06		S
isolation_enable_condition	2014.09	2017.06		S
power_down_function	2014.09	2017.06		S
switch_pin	2014.09	2017.06		S
is_analog	2014.09	2017.06		S

D

Virtuoso Power Manager Forms

This appendix describes the Virtuoso Power Manager forms.

Add 1801 File Binding

Adds a cell and binds it to its existing 1801 power intent file. For details, refer to Miscellaneous Settings.

Field Name	Description
Cell	The cell to be registered for binding.
1801 File	The name of the 1801 file for binding.

Add Device

Registers the device type from a specific library and cell. For details, refer to Device and Cell Registration.

Field Name	Description
Device Type	The types of devices available in the design.
Library	The library of the selected device type.
Cells	The cell of the selected device type.

Add Port Attributes

Adds the attributes of a port. These attributes override those extracted by the tool during supply tracing. For details, refer to Port Attributes Registration.

Field Name	Description
Ports	The list of the ports to be added.
Exclude Ports	The ports to be excluded during registration.
Driver Supply Set	The driver supply set.
Receiver Supply Set	The receiver supply set.
Analog Port	Identifies the port or list of ports as analog. These have an associated `is_analog` attribute in the extracted 1801 or exported macro Liberty model.

Unconnected Port	Identifies the port or list of ports as unconnected. These have an associated `is_unconnected` attribute in the extracted 1801 file or exported macro Liberty model.
Feedthrough Port	Identifies the port or list of ports as feedthrough. These have an associated `short` attribute in the extracted 1801 or exported macro Liberty model.

Add Power Domain

Registers power domains corresponding to the supply sets with the required attributes. This includes supply sets associated with the power domain along with the elements that constitute the power domain. For details, refer to Supply Set and Power Domain Registration.

Field Name	Description
Power Domain	Specifies the name of the power domain to be registered.
Primary Supply	Specifies the supply sets associated with the power domain.
Elements	Specifies the elements of the power domain.
Exclude Elements	Specifies the elements to be excluded from the power domain.
Include Scope	Specifies whether the power domain should a top-level scope. All the elements at the current level and the levels below fall in the scope of this power domain.

Add Supply Nets

Lets you specify the supply nets for the Power Manager setup. For details, refer to Supply Nets Registration.

Field Name	Description

Filter	Filters the list of supply nets to limit the search scope.
Add Nets	Adds the user-specified supply nets.
Net Type	Specifies the type of the supply or ground net.
Voltage	Specifies the voltage value of the supply nets.
External Switch Type	Specifies whether the supply net is externally switchable.

Add Supply Sets

Adds supply sets with the required attributes. This includes registering the supply nets along with ground nets. For details, refer to Supply Set and Power Domain Registration.

Field Name	Description
Supply Set	The various types of nets based on the substrate.
Power	
Ground	
Pwell	
Nwell	
DeepPwell	
DeepNwell	

Add Supply State

Adds you specify the supply states along with the voltage values. For details, refer to In-Design Checks.

Field Name	Description

Get Default Supply State	Populates default supply states. The default supply state has all the registered supply voltages from the *Registered Supply Nets* table and populates all of them in the form with value as 'OFF'.
Supply State	A user-defined supply state name. Each supply state consists of combinations of different supply voltages and their corresponding voltage values.
Voltage	The supply voltages that are registered in the setup on the Supply Nets tab or new.
Value	The supply voltage values corresponding to each supply voltage. This information is derived from the supply voltage values registered in the *Supply Nets* table in the setup. Each supply voltage comprises an OFF voltage value, which means that there is no deterministic voltage value for that supply in the current supply state. You can also specify a user-defined supply voltage value for supply voltage.

Generate Signal Information

Lets you specify the file that contains the signal information for the given cellview. For details, refer to Generating Signal Information.

Field Name	Description
Library	Specifies the library for which the signal information is to be generated.
Cell	Specifies the cell for which the signal information generated.
View	Specifies the view name for which the signal information generated.
Signal Information File	Specifies the name of the file that contains the signal information.

Load Violations

Lets you specify the cellview and review its design violations in the Annotation Browser. For details, refer to In-Design Checks.

Field Name	Description
Library	Specifies the library of the design that has violations.
Cell	Specifies the cell of the design that has violations.
View	Specifies the view name of the design that has violations.
Violations File	Specifies the name of the file that will be uploaded to view the design violations.

Run In-Design Checks

Runs In-Design checks and generates the log and violations files. For details, refer to Running In-Design Checks.

Field Name	Description
Library	The library of the cellview to be checked.
Cell	The cell of the cellview to be checked.
View	The view name of the cellview to be checked.
Log File	The name of the log file to be generated at the end of checks.
Violations File	The name of the file that will contain the design violations at the end of checks.