cadence®

Common Power Format Language Reference

Version 2.0
October 2019

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

Contents

8
General CPF Commands

Alphabetical List of Commands

U

Preface

About This Manual

This manual is a language reference for users of the Common Power Format (CPF). This manual explains concepts introduced by CPF, describes the format specifics, and gives a detailed explanation for each CPF command.

Additional References

For information on what is new or changed in CPF version 1.1 see *What's New in Common Power Format*.

For more information on the usage of CPF, refer to the *Common Power Format User Guide*.

The following sources provide more infromation on Tcl:

- http:// en.wikipedia.org/wiki/Tcl
- TclTutor, a computer aided instruction package for learning the Tcl language: http://www.msen.com/~clif/TclTutor.html.
- TCL Reference, *Tcl and the Tk Toolkit*, John K. Ousterhout, Addison-Wesley Publishing Company

Reporting Problems or Errors in Manuals

The Cadence® Help online documentation, lets you view, search, and print Cadence product documentation. You can access Cadence Help by typing `cdnshelp` from your Cadence tools hierarchy.

Contact Cadence Customer Support to file a CCR if you find:

- An error in the manual
- An omission of information in a manual
- A problem using the Cadence Help documentation system

Customer Support

Cadence offers live and online support, as well as customer education and training programs.

Cadence Online Support

The Cadence® online support website offers answers to your most common technical questions. It lets you search more than 40,000 FAQs, notifications, software updates, and technical solutions documents that give you step-by-step instructions on how to solve known problems. It also gives you product-specific e-mail notifications, software updates, case tracking, up-to-date release information, full site search capabilities, software update ordering, and much more.

For more information on Cadence online support go to:

http://support.cadence.com

Other Support Offerings

- **Support centers**—Provide live customer support from Cadence experts who can answer many questions related to products and platforms.
- **Software downloads**—Provide you with the latest versions of Cadence products.
- **Education services**—Offers instructor-led classes, self-paced Internet, and virtual classroom.
- **University software program support**—Provides you with the latest information to answer your technical questions.

For more information on these support offerings go to:

http://www.cadence.com/support

Documentation Conventions

To aid the readers understanding, a consistent formatting style has been used throughout this manual.

The list below describes the syntax conventions used for the CPF constraints.

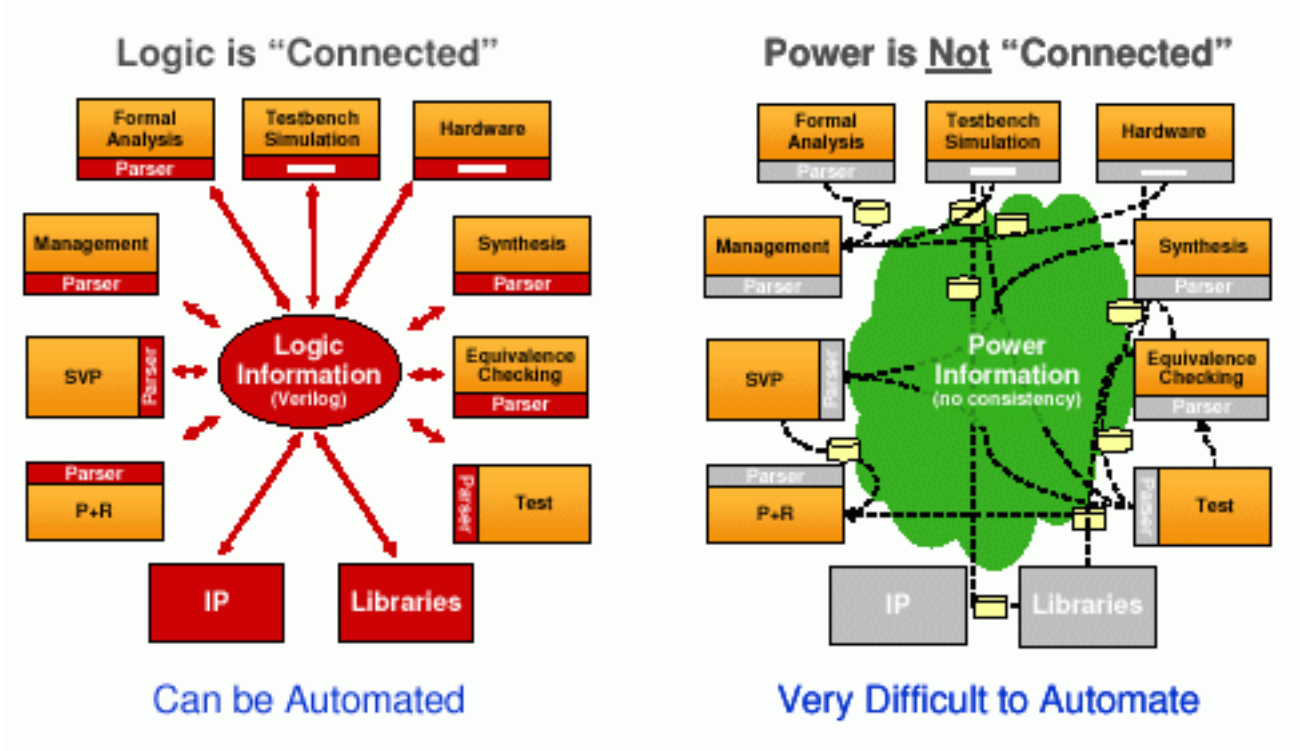
`literal`	Nonitalic words indicate keywords that you must type literally. These keywords represent command or option names. An option always contains a leading minus character (-).
`arguments and options`	Words in italics indicate user-defined arguments or options for which you must substitute a value. A value can be one of the following: ■ string ■ integer or floating number ■ object(s) reference ■ a list of values
|	Vertical bars (OR-bars) separate possible choices for a single argument.
`[ ]`	Brackets denote options. When used with OR-bars, they enclose a list of choices from which you can choose one.
`{ }`	Braces denote arguments and are used to indicate that a choice is required from the list of arguments separated by OR-bars. You must choose one from the list `{ argument1 | argument2 | argument3 }`
`{ }`	Braces in bold-face type must be entered literally.
`...`	Three dots (...) indicate that you can repeat the previous argument. If the three dots are used with brackets (that is, `[argument]...`), you can specify zero or more arguments. If the three dots are used without brackets (`argument...`), you must specify at least one argument, but can specify more.
#	The pound sign precedes comments.

1

Introducing the Common Power Format

The shift in the use of chips to consumer applications and the change in the latest process technologies have made power one of the primary design criteria for a majority of the chips worldwide. However, the industry's design infrastructure has not evolved at the same pace. Figure1-1 shows the mature state of the infrastructure for functional designs versus the chaotic state of the infrastructure for designs using advanced low power design techniques.

Figure 1-1 Comparison of State of Infrastructures for Functional Designs and Advanced Low Power Designs



To accomplish an industry-wide solution for this industry-wide problem, every effort was made to use an open and inclusive approach to create a complete and well architected solution.

The lack of support in the infrastructure for designs using advanced low power design techniques has resulted in a gap between the design techniques needed to control power dissipation and the ability of the design environment to support those techniques in a safe and efficient manner. The **C**ommon **P**ower **F**ormat has been architected to supply the infrastructure needed to support the state of the art in low power design styles and techniques.

The requirements for the Common Power Format were created using a wide range of viewpoints and with a broad range of applications in mind:

■ Semiconductor manufacturing equipment	■ High-end graphics processing
■ Semiconductor manufacturing (foundry)	■ Cell phone design
■ Library provider	■ Processor design
■ IDM (system design through silicon manufacturing) consumer, computing, networking	■ Intellectual Property (core processors & peripherals)
■ EDA	■ Automotive

The broad participation in creating the requirements specification ensured the architecture of a comprehensive solution that would be complete in nature. Some primary requirements are:

- **Easy to adopt**—to overcome cost, time and risk deployment issues.
- **Incremental** to existing infrastructure—overlay on top of methods in place.
- **Non-intrusive** to existing practices, methodologies and flows
- **Serves IP/re-use** methodologies with a minimal incremental effort
- **Consolidated** view of the power strategy for a design into a single entity
- **Comprehensive** in capabilities to support the most advanced existing low power design techniques, across the entire continuum of design automation.
- **Extensible** to new low power design techniques and to broader design flow scope (up to system-level and into analog mixed signal in particular).

A bottom-up analysis has led to support for a digital RTL to sign-off solution. Although limited in scope, the solution is broad in terms of design automation technology inclusion:

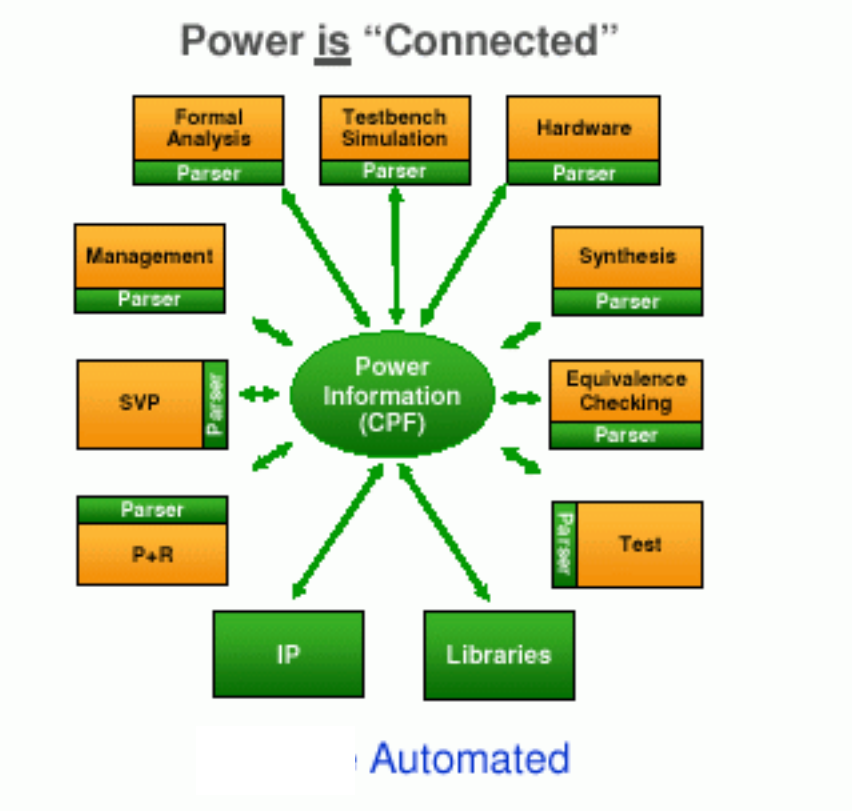
■ RTL/gate simulation	■ Physical synthesis / placement
■ Hardware simulation acceleration	■ Clock tree synthesis
■ Hardware emulation	■ Power grid design
■ Formal analysis	■ Power integrity analysis
■ Design analysis & rule checking	■ Design for Test
■ Formal verification	■ Automatic test pattern generation
■ Synthesis & optimization	■ Constraint generation
■ Floorplanning	■ Constraint verification
■ Silicon virtual prototyping	■ Design project management
■ Power analysis	■ Design IP

Adopting the Common Power Format into standard design flows will have fundamental benefits to those that use it along with industry leading tool solutions. It

- Enables RTL functional verification to validate power related operation
- Guarantees higher design quality with fewer functional failures
- Reduces risk in applying state-of-the-art low power design techniques
- Increases productivity and reduced cost of using those power saving methods

Figure 1-2 shows the benefit of the Common Power Format in the design flow.

Figure 1-2 Benefit of the Common Power Format on the Design Flow



2

Terminology

Design Objects

Design objects are objects named in the description of the design which can be in the form of RTL files or a netlist. Design objects can be referenced by the CPF commands.

Design

The top-level module.

Followpins

Routing structures in the standard cells that allow routing of power and ground nets in a standard cell row through abutting of cells. Because the power and ground pins in the cells are aligned, the power and ground routing “follows the pins.”

Instance

An instantiation of a module or library cell.

- Hierarchical instances are instantiations of modules.
- Leaf instances are instantiations of library cells.

Module

A logic block in the design.

Net

A connection between instance pins and ports.

A *logical* net is a connection between pins or ports in the same module.
A *logical net segment* is a connection between one driver and one load within the same module.
A logical net with multiple fanouts has multiple (logical) net segments.

A *physical* net is a connection between an leaf-level driver and one or more leaf-level loads. All logical nets that are electrically connected make up one physical net.

Note: In this document net refers to logical net.

Pad

An instance of an I/O cell, also referred to as a pad cell. The cell typically has one or more pins that must be connected to the package pins of a chip. Such pins are referred to as pad pins or pad ports.

Pin

An entry point to or exit point from an instance or library cell.

Port

An entry point to or exit point from the design or a module.

CPF Objects

CPF objects are objects that are being defined (named) in the CPF constraint file. CPF objects can be referenced by the CPF commands.

Analysis View

A view that associates an operating corner (or another lower-level analysis view in a hierarchical flow) with a power mode for which timing constraints were specified.

The set of active views represent the different design variations (MMMC, that is, multi-mode multi-corner) that will be timed and optimized.

Base and Derived Power Domains

A power domain whose primary power supply provides power to another power domain through a power switch network is called a ***base*** domain.

A power domain that derives its power from another power domain through a power switch network is called the ***derived*** domain.

Power domains can have multiple base domains.

A power domain can be a base domain and be a derived power domain from another power domain.

Isolation Rule

Defines the location and type of isolation logic to be added and the condition for when to enable the logic.

Level Shifter Rule

Defines the location and type of level shifter logic to be added.

Library Group

A list of libraries characterized for two or more operating conditions. All libraries in a group must have the same cells. A library group can be used in a DVFS design to interpolate power or timing data for any operating conditions.

Library Set

A set (collection) of libraries or library groups. By giving the set a name it is easy to reference the set when defining nominal conditions or operating corners.

The same library set can be referenced multiple times by different operating corners.

Mode

A static state of a design that performs one or more intended design functions. Typically, it is determined by the states of memory elements, states of power domains, and signal values.

Mode Transition

Defines when the design transitions between the specified power modes.

Nominal Operating Condition

A typical operating condition under which the design or blocks perform. An operating condition is determined by the voltages of all power supplies applied to a power domain, including the power voltage, ground voltage and the body bias voltage for PMOS and NMOS transistors. Depending on the technology used, this set of voltages determines whether the state of a power domain is on, off or in standby mode.

Operating Corner

A specific set of process, voltage, and temperature values under which the design must be able to perform.

Power Design

A unique power structure that can be associated with either a top design, or with one or more logic modules.

When a power design is associated with multiple logic modules, it allows the same power intent specification to be applied to instances of different modules.

On the other hand, a single logic module can have multiple power designs associated with it, allowing different power intent specifications to be applied to instances of a single design entity.

Power Domain

A collection of instances, pins and ports that can share the same power distribution network.

At the *physical level* a power domain contains

- A set of power supply nets including a single pair of primary power and ground nets and optionally bias power and/or ground nets.
- A set of cells with a single power and a single ground rail connecting to the primary power and ground nets
- A set of special gates such as level shifter cells, state retention cells, isolation cells, power switches, always-on cells, or multi-rail hard macros (such as, I/Os, memories, and so on) with multiple power and ground rails

  At least one pair of the power or ground rails in these special gates or macros must be connected to the primary power and ground nets of the power domain.

Note: Two or more power domains can have the same set of power and ground nets.

At the *logic level* a power domain contains

- A set of logic gates that correspond to the (regular) physical gates of this power domain
- A set of special gates such as level shifter cells, state retention cells, isolation cells, power switches, always-on cells, or multi-rail hard macros (such as, I/Os, memories, and so on) that correspond to the physical implementation of these gates in this power domain

At the *RTL level* a power domain contains

- The computational elements (operators, process, function and conditional statements) that correspond to the logic gates in this power domain

Note: See also Base and Derived Power Domains and Secondary Power Domain.

Power Mode

A static state of a design in which each power domain operates at a specific nominal condition.

Power Mode Control Group

A set of power domains with an associated set of power modes and mode transitions that apply to this group only. A power mode group can contain other power mode groups.

Power Source Domain

A power domain that models the power source for the primary supply or the body bias supply of other domains.

Power Switch Rule

Defines the location and type of power switches to be added and the condition for when to enable the power switch.

Secondary Power Domain

A power domain `X` is a ***secondary*** power domain of a special low power instance if the primary power supply of domain `X` provides the power supply to the non-switchable (secondary) power and (or) ground pins of the instance.

State Retention Rule

Defines the registers or regular flip-flop and latch instances to be replaced with state retention flip-flops and latches and the conditions for when to save and restore their states.

Virtual Port

A port that does not exist in the definition of a module before implementation, but that will be needed for the control signals of the low power logic such as isolation logic, state-retention logic, and so on. After implementation, these ports are added to the module definition in the netlist.

Virtual Power Domain

Power domain without instances and with no power and ground nets defined.

Special Library Cells for Power Management

Always On Cell

A special cell located in a switched-off domain that has secondary power or ground pins in addition to the primary power and ground pins. As long as the power supply to the secondary power or ground pins is on, the cell function does not change.

A special case of a global cell.

Global Cell

A special cell that has secondary power or ground pins in addition to the primary power and ground pins (*followpins*). In some cell designs, when the primary power or ground are switched off, the cell function can be different from the normal function when the primary power and ground are on. Also, in some cases, the cell can also have isolation logic built in at the cell input pins.

Examples of global cells are traditional always-on cells and other special low-power cells such as state retention cells and dual-rail isolation cells.

Isolation Cell

Logic used to isolate signals between two power domains where one is switched on and one is switched off.

The most common usage of such cell is to isolate signals originating in a power domain that is being switched off, from the power domain that receives these signals and that remains switched on.

Level Shifter Cell

Logic to pass data signals between power domains operating at different voltages.

Power Clamp Cell

A special diode cell to clamp a signal to a particular voltage.

Power Switch Cell

Logic used to connect and disconnect the power supply from the gates in a power domain.

State Retention Cell

Special flop or latch used to retain the state of the cell when its primary power supply is shut off.

3

Understanding the CPF Format

Introduction

The Common Power Format (CPF) is a strictly Tcl-based format.

The CPF file is a power specification file. This implies that the functionality of the design does not change when sourcing a CPF file. The CPF file complements the HDL description of the design and can be used throughout the design creation, design implementation, and design verification flow.

Objects

The CPF file contains two categories of objects:

- Design Objects are objects that already exists in the description of the design.
- CPF Objects are objects that are created in the CPF file.

Object Names

Design Objects

The design object name must be a valid SystemVerilog or VHDL name. If a CPF file is written for RTL, use the RTL object names. if a CPF is written for a gate-level netlist, use the gate-level netlist object names.

Note: In VHDL, object names in the format of extended identifier (VHDL LRM,13.3.2) are not supported.

See Object References for more information on referencing design objects.

CPF Objects

CPF objects are created by CPF commands. The string that follows the `-name` option in a CPF command identifies the CPF object that is created. For example,

```
create_power_mode -name PM1 ...
```

This command creates a power mode `PM1` in the current scope (part of the design that is visible).

A CPF object can have the same name as a design object, but must have a unique name within the scope.

A CPF object name cannot contain the hierarchy delimiter character.

CPF object names can contain

- any sequence of letters (of the alphabet)
- digits
- the underscore (_)
- the period (.)

  Note: The period (.) can only be used for a nominal condition name and the operating corner.

See Object References for more information on referencing CPF objects inside and outside the current scope.

Object Lists

Lists of objects must be enclosed in braces. CPF files can contain simple and complex lists.

A simple list is a list of *single* design objects or CPF objects.

A complex list can be a

- List of lists

  For example: `{{iso_en PCM/ISO} {restore PCM/WAKE}}`

- List of compound objects

  For example: `{PD1@high_max PD2@high_min PD3@low_max}`

  A compound object is created by joining two CPF objects with the at sign (@) character.

Empty Lists

In some cases, the object list specified with a command option can be empty. For example,

```
create_isolation_rule -name ISO1 -from PD1 -pins [find_design_objects \
-object_type port A*]
```

If the design has no ports matching the pattern A*, the `find_design_objects` command returns an empty list. In this case, the isolation rule does not apply to any ports in the design. This is equivalent to specifying:

```
create_isolation_rule -name ISO1 -from PD1 -pins {}
```

Escape Character

The Tcl escape character is the backslash character (\). Use it to escape special characters that have special meaning to the Tcl interpreter, such as square brackets.

Note: The escape character is not required when the special character is contained within curly braces or double quotes.

Hierarchy Delimiter

The supported hierarchy delimiter characters are

- period (.) — *default*
- slash (/)
- colon (:)

The hierarchical delimiter can be specified using the set_hierarchy_separator command. See Information Inheritance for more information on the scope sensitivity of this command.

This character only has this special meaning in object names. An escaped hierarchy delimiter character loses its meaning as a hierarchy delimiter.

The semantics of building an object reference using the hierarchical separator are specified in Object References on page 121.

In the following rule example, the period is the default hierarchy delimiter. The first period is escaped, while the second period is not. Consequently the second period acts as hierarchy delimiter, and instance `c[0]` is considered to belong to hierarchical instance named `block.xxx`.

```
create_state_retention_rule -name ret -instances {block\.xxx.c[0]}
```

Note: The hierarchy delimiter is also the pin delimiter when referencing a pin object.

Wildcards

You can specify multiple objects of the same type by including wildcards. Wildcards are expanded only on objects within the current scope.

- * matches zero or more characters
- ? matches a single character

Important

Wildcard characters can represent bus delimiters, but they do *not* represent the hierarchical separator.

For example, c*[3] can represent `c1[3]` and `c1[0][3]`, while `a*/b` can represent `a1/b` and `a2/b`, but not `a/x/b` or `a/x/y/b` (assuming / is used as the hierarchical separator).

Note: In rule specifications, wildcards are considered to specify the targeted design objects *explicitly* (by expanding the string to a list). See Different Categories of Rules.

Bus Delimiters

The default bus delimiters are the square brackets ([ ]).

Because the square brackets ([ ]) are reserved characters in Tcl syntax, you can do one of the following for bus bit names:

- Enclose the bus bit name in braces

  ```
  -pins {a[0]}
  ```

- Escape the open square bracket ([) and the closing square bracket (]) when you use these characters as bus delimiters

  ```
  -pins a\[0\]
  ```

The square brackets only have this special meaning in object names.

When the entire object name is escaped, the square brackets lose their meaning as bus delimiters. For example, the Verilog names `a[0]` and `\a[0]` are not equivalent. Only `a[0]` represents bit 0 of bus `a`, while `\a[0]` is a scalar object name. To avoid Tcl errors for the scalar object name in CPF, enter it as either

- `a\\[0\\]`
- `{ a\[0\] }`

The recommended format for a list of objects is: `{{a\[0\]} {a\[1\]}}`

Range Specification

To specify a range (multiple bits of a bus or of a register array), use the bus delimiters and the colon (:). For example:

```
a[2:7]
b[6:3]
c_reg[4:2]
```

To specify all pins of a bus pin or bus port, use the bus pin or bus port name without a range.

Individual Registers Names

For a CPF file written for an RTL design, the register names are based on

- A base name
- (optional) A bit select appended to the base name.

Examples of register names in CPF: `register_A`, `register_A[0]`

The format of a register name in **RTL** and the corresponding flip-flop or latch instance names in the **netlist** can be different. When reading in a CPF file written for RTL together with a gate-level netlist, you need to specify how the base name and bit information are represented in the gate-level netlist. This is explained in the following sections.

Specifying the Representation of the Base Name

➤ To specify the suffix that is appended to the base name of a flip-flop or latch instance in the netlist, use the set_register_naming_style command.

The `set_register_naming_style` command expects a string with the following format:

`string%s`

The default format is: `_reg%s`

See Information Inheritance for more information on the scope sensitivity of this command.

The following rules apply:

- An instance name is always started with the base name.
- The suffix is appended to the base name to form the instance name, according to the format specified in the string.

- If the corresponding RTL register is an array, `%s` represents the bit information (see also Specifying the Representation of the Bits).

Specifying the Representation of the Bits

➤ To specify how the bit information of a flip-flop or latch instance is represented in the netlist, use the set_array_naming_style command.

The `set_array_naming_style` command expects a string with the following format:

[*character*]%d[*character*]

The default format is: `\[%d\]`

See Information Inheritance for more information on the scope sensitivity of this command.

For example, this option can have values such as:

```
<%d>,\[%d\], _%d_, _%d
```

The following rules apply:

- A suffix is generated for each dimension, according to the format specified in this string.
- The %d represents an index of a certain dimension.

All pieces of the suffix are concatenated, from the highest dimension to the lowest dimension, to form a combined suffix for multi-dimensional arrays.

Examples

Assume the following RTL input:

```
reg a;
reg [3:2] b;
reg [5:4][3:2] c;
```

- If you specify the following commands:

```
set_register_naming_style %s
set_array_naming_style _%d
```

The corresponding instance names in the netlist are expanded as follows:

```
a
b_2
b_3
c_4_2
c_4_3
c_5_2
```

```
c_5_3
```

- If you specify the following commands:

```
set_register_naming_style _reg%s
set_array_naming_style _%d_
```

The instance names are expanded as follows:

```
a_reg
b_reg_2_
b_reg_3_
c_reg_4__2_
c_reg_4__3_
c_reg_5__2_
c_reg_5__3_
```

- If you specify the following commands:

```
set_register_naming_style %s
set_array_naming_style \[%d\]
```

The instance names are expanded as follows:

```
a
b[2]
b[3]
c[4][2]
c[4][3]
c[5][2]
c[5][3]
```

Important

Because the square brackets represent command substitution in the Tcl language, you need to either escape the entire instance name or escape each bracket character and enclose the entire name in braces, if you want to reference these instance names in CPF commands. The following are equivalent:

```
{c[4][2]}
and
c\[4\]\[2\]
```

Expressions

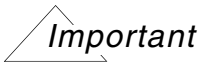
In this document, all expressions refer to SystemVerilog Boolean expressions. The current version only supports the following operators for Boolean expressions:

Operator	Description
~	bit-wise negation
&	bit-wise AND
|	bit-wise OR
^	bit-wise XOR
!	logical negation
&&	logical AND
||	logical OR

All operators associate left to right. When operators differ in precedence, the operators with higher precedence apply first. In the table above, the operators are shown in order of precedence.

Unless otherwise specified, operands can be one of the following:

- A port
- An instance pin
- A library cell pin



- You can use parentheses () to change the operator precedence.
- Pin names in expressions cannot represent buses.
- If a design object name contains a Boolean operator, you must escape the Boolean operator.

Units

To specify the power unit, use the set_power_unit command. The default power unit is `mW`.

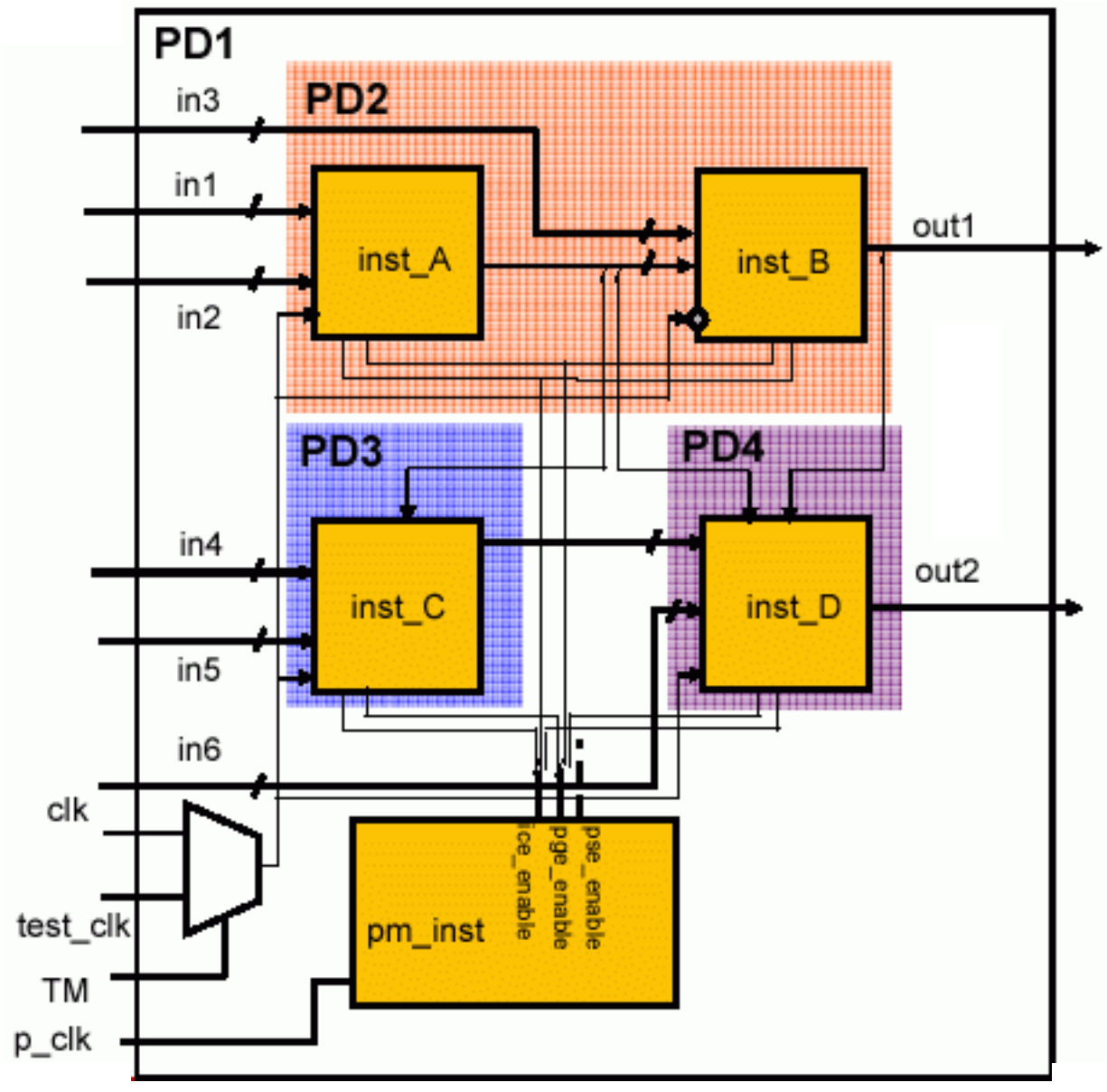
To specify the time unit, use the set_time_unit command. The default time unit is `ns`.

All voltage values must be specified in volt (V).

Example

Consider the example design shown in Figure 3-1 on page 33.

Figure 3-1 Example Design for CPF



The design has four domains:

- The top-level of the design and hiearchical instance `pm_inst` belong to the default domain `PD1`

- Hierarchical instances `inst_A` and `inst_B` belong to power domain `PD2`

  Hierarchical instance `inst_B` is an instantiation of an IP block. Its power domain is mapped to power domain `PD2`.

  The IP is designed to be used either in a switchable or non-switchable top design. If it is used in a switchable top design, all of the registers of `inst_B` need to be implemented as state retention flops.

- Hierarchical instance `inst_C` belongs to power domain `PD3`
- Hierarchical instance `inst_D` belongs to power domain `PD4`

Table 3-1 on page 34 shows the static behavior (voltage) for each power domain in each of the modes.

Note: A voltage of 0.0V indicates that the power domain is off.

Table 3-1 Static Behavior

Power Mode	Power Domain			
	PD1	PD2	PD3	PD4
PM1	1.2V	1.1V	1.2V	1.0V
PM2	1.2V	0.0V	1.2V	1.0V
PM3	1.2V	0.0V	0.0V	1.0V
PM4	1.0V	0.0V	0.0V	0.0V

The power manager (`pm_inst`) generates three sets of control signals to control each power domain. These signals are available on pins `pse_enable`, `ice_enable` and `pge_enable`.

Table 3-2 Signals Controlling the Power Domains

Power Domain	Control Signals		
	power switch	isolation cell	state retention cell
PD1	no control signal	no control signal	no control signal
PD2	pse_enable[0]	ice_enable[0]	pge_enable[0]
PD3	pse_enable[1]	ice_enable[1]	pge_enable[1]

Power Domain	Control Signals		
	power switch	isolation cell	state retention cell
PD4	pse_enable[2]	ice_enable[2]	pge_enable[2]

CPF File of IP

```
# Define IP mod_B, file name IPB.cpf
#----------------------------------
set_design power_design_B -modules mod_B
create_power_domain -name PDX -default
create_state_retention_rule -name RETB1 -domain PDX
end_design power_design_B
```

CPF File of Top Design

```
# Define top design
#------------------
set_design top
# Set up logic structure for all power domains
#---------------------------------------------
include IPB.cpf
create_power_domain -name PD1 -default
create_power_domain -name PD2 -instances {inst_A} \
-shutoff_condition {!pm_inst.pse_enable[0]} -base_domains PD1
create_power_domain -name PD3 -instances inst_C \
-shutoff_condition {!pm_inst.pse_enable[1]} -base_domains PD1
create_power_domain -name PD4 -instances inst_D \
-shutoff_condition {!pm_inst.pse_enable[2]} -base_domains PD1
set_instance inst_B -domain_mapping { PDX PD2 } -design mod_B
# Define static behavior of all power domains and specify timing constraints
#---------------------------------------------------------------------------
set_instance inst_B -design power_design_B -domain_mapping { PDX PD2 }
create_nominal_condition -name high -voltage 1.2
create_nominal_condition -name medium -voltage 1.1
create_nominal_condition -name low -voltage 1.0
create_power_mode -name PM1 -domain_conditions {PD1@high PD2@medium PD3@high \
PD4@low}
update_power_mode -name PM1 -sdc_files ../SCRIPTS/cm1.sdc \
-activity_file ../SIM/top_1.tcf -activity_file_weight 1
create_power_mode -name PM2 -domain_conditions {PD1@high PD3@high PD4@low}
update_power_mode -name PM2 -sdc_files ../SCRIPTS/cm2.sdc
create_power_mode -name PM3 -domain_conditions {PD1@high PD4@low}
create_power_mode -name PM4 -domain_conditions {PD1@low}
# Set up required isolation and state retention rules for all domains
#--------------------------------------------------------------------
create_state_retention_rule -name sr1 -domain PD2 \
-restore_edge {!pm_inst.pge_enable[0]}
create_state_retention_rule -name sr2 -domain PD3 \
-restore_edge {!pm_inst.pge_enable[1]}
```

```
create_state_retention_rule -name sr3 -domain PD4 \
-restore_edge {!pm_inst.pge_enable[2]}
create_isolation_rule -name ir1 -from PD2 \
-isolation_condition {pm_inst.ice_enable[0]} -isolation_output high
create_isolation_rule -name ir2 -from PD3 \
-isolation_condition {pm_inst.ice_enable[1]}
create_isolation_rule -name ir3 -from PD4 \
-isolation_condition {pm_inst.ice_enable[2]}
create_level_shifter_rule -name lsr1 -to {PD1 PD3}
end_design
```

4

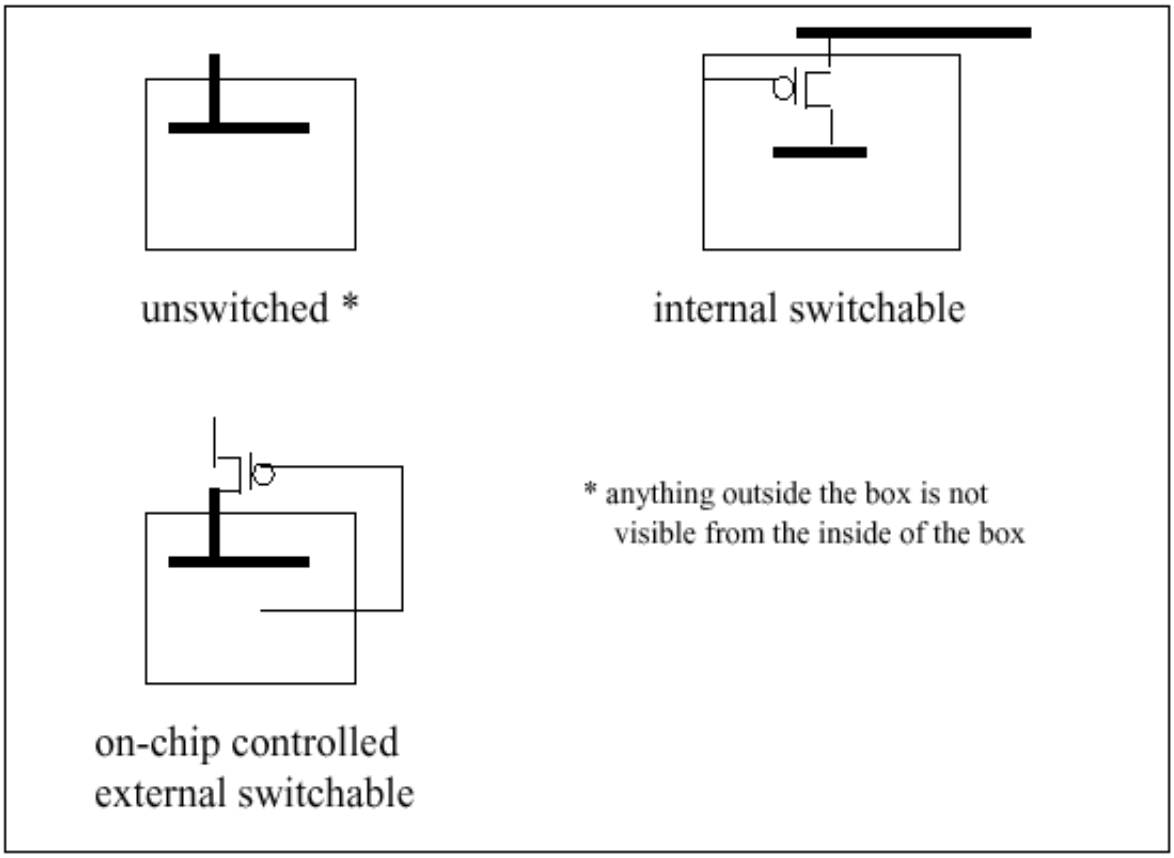
Power Domains and Modes

Power Domains

Power Domain Categories

CPF considers three categories of power domains as shown in Figure 4-1.

Figure 4-1 Categories of Power Domains



Designers must accurately describe the design intention by correctly specifying the shutoff conditions. Implementers must correctly describe the power and ground network and indicate if there are external shutoff conditions. Verification tools should be able to check if the implementation matches the design intent.

unswitched domain

This domain is never powered down by controls in the scope in which it is created.

To model an unswitched domain

- Specify the create_power_domain command without the `-shutoff_condition` or the `-external_controlled_shutoff` options.
- If you need to create its power and ground nets in CPF, create them without any external shutoff conditions.

Tip

An unswitched power domain can be part of a power mode definition where the domain is associated with a nominal condition of state `off`, indicating that the domain can be shut off externally.

```
create_power_domain -name PD1 -instances {...}
create_power_domain -name PD2 -instances {...}
create_power_mode -name PM1 -domain_conditions {PD1@on PD2@on}
create_power_mode -name PM2 -domain_conditions {PD1@on PD2@off}
```

In the previous example, `PD1` is always on in current scope, but `PD2` can be switched off externally.

internal switchable domain

This domain can be powered down by an on-chip power or ground switch. This domain gets its power supply from another domain through a power switch network.

To model an internal switchable domain

- Specify the condition that causes the power domain be powered down through the `-shutoff_condition` option of the create_power_domain command.
- Specify the domain from which the switchable domain (or the power switch) gets its power supply through the `-base_domains` option of the `create_power_domain` command.
- (optional) For physical implementation, define the on-chip power or ground switch using a power switch rule associated with the internal switchable domain (see Chapter 6, "Precedence and Semantics of the Rules," for more information).

  For an internal switchable domain, the power switch rules associated with the power domain (specified with the `-enable_condition_`*x* option of the update_power_switch_rule command) determine how the shutoff behavior for the domain will be implemented. The verification tools should check the consistency between the shutoff condition defined for the power domain and the shutoff condition derived from all associated power switch rules.

on-chip controlled external switchable domain

This domain can be powered down by an external power switch (outside of the chip), but the external switch is controlled by signals on the chip.

To model an on-chip controlled external switchable domain

➤ Specify the condition that causes the power domain be powered down through the `-shutoff_condition` option together with the `-external_controlled_shutoff` option of the create_power_domain command.

Note: Since the power (or ground) switch is not part of the chip, it cannot be described through a power switch rule.

For an on-chip controlled external switchable domain, the implementation of the shutoff behavior for the domain is determined by the external shutoff condition specified for the primary power and ground nets of the power domain (defined in the update_power_domain command). The verification tools should check the consistency between the shutoff condition of the power domain and the external shutoff condition of the primary power and ground nets.

If an external shutoff condition is specified for the primary power and (or) ground net of an on-chip controlled external switchable power domain, it must match the shutoff condition specified for that domain in one of the following ways:

- If an external shutoff condition is defined for only the primary power net, it must be identical to the shutoff condition of the domain.
- If an external shutoff condition is defined for only the primary ground net, it must be identical to the shutoff condition of the domain.
- If an external shutoff condition is defined for both the primary power and ground nets, the domain shutoff condition must be identical to the Boolean OR of the external shutoff condition definition of the power and ground nets.

Example

If power domain `Z` is specified with shutoff condition `!A|!B` and with the `-external_controlled_shutoff` option, power domain `Z` is considered on-chip controlled external switchable.

If `VDD` and `VSS` are the primary power net and primary ground net for the domain, respectively, then the following specifications are valid for this design intent:

- `VDD` is specified with external shutoff condition `!A|!B`. `VSS` has no external shutoff condition.
- `VSS` is specified with external shutoff condition `!A|!B`. `VDD` has no external shutoff condition.
- `VDD` is specified with external shutoff condition `!A` and `VSS` is specified with external shutoff condition `!B`.
- `VDD` is specified with external shutoff condition `!B` and `VSS` is specified with external shutoff condition `!A`.

Base and Derived Power Domains

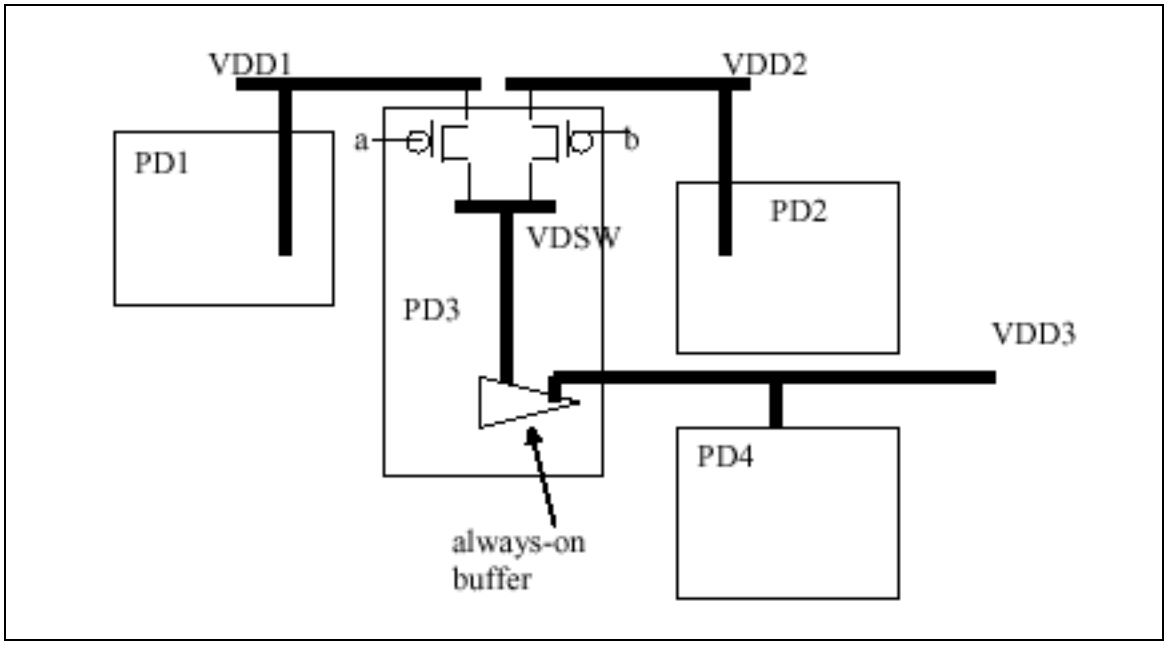
A power domain `X` is a ***base*** power domain of power domain `Y` if the primary power and ground nets of power domain `X` provide the power supply to domain `Y` through some power switch network. Domain `Y` is called the ***derived*** domain of domain `X`.

A derived power domain connects through some sort of device, e.g. a header, footer, or regulator, to one or more base power domains. The shutoff condition for a derived power domain is defined in terms of ports and pins inside this scope (-shutoff_condition) and from which power domain it is derived (-base_domains). A derived power domain is considered to be off whenever all of its base power domains are off.

Example

Power domain `PD3` in Figure 4-2 has two base power domains, `PD1` and `PD2`.

Figure 4-2 Sample Design



For an internal switchable domain, an error should be given if the base domain definition does not match the power switch rule definition of the domain. For example, the external power net of a power switch rule must be the primary power net of one of the base domains defined in the `-base_domains` option.

For the example in Figure 4-2 on page 44, the following CPF is valid:

```
create_power_domain  -name PD1 ...
create_power_domain  -name PD2 ...
create_power_domain  -name PD3  -shutoff_condition { a & b } \
  -base_domains {PD1 PD2} ...
create_power_domain  -name PD4 ...
update_power_domain  -name PD1  -primary_power_net VDD1
update_power_domain  -name PD2  -primary_power_net VDD2
update_power_domain  -name PD3  -primary_power_net VDSW
create_power_switch_rule -name r1 -domain PD3 -external_power_net VDD1
update_power_switch_rule -name r1 -enable_condition_1 a
create_power_switch_rule -name r2 -domain PD3 -external_power_net VDD2
update_power_switch_rule -name r2 -enable_condition_1 b
```

In the above example, the external power nets of the switch rules of `PD3` are `VDD1` and `VDD2` which matches the primary power nets defined for the base domains, `PD1` and `PD2`.

It is also possible to have an unswitched domain with a base domains defined. For example, if the two control signals `a` and b will never be high at the same time, domain `PD3` is an unswitched domain and should be created without a shutoff condition. The power switches are just a way to implement dynamic voltage scaling control. In this case, `PD3` still has `PD1` and `PD2` as its base domains by definition.

Primary and Secondary Power Domains of Instances

A power domain `X` is a ***secondary*** power domain of a special low power instance if the primary power and ground nets of the domain `X` provide the power supply to the secondary power and (or) ground pins of the instance.

A power domain `Y` is a ***primary*** power domain of a standard cell instance if the primary power and ground nets of domain `Y` provide the power supply to the primary power and ground pins (follow-pins) of the cell.

The secondary power domain definition for special low power logic is handled by corresponding `create_xx_rule` commands. CPF also provides a special command to give the user the flexibility to specify the expected secondary power domain of any special low power standard cell instances. If the secondary domain is neither specified in the `identify_secondary_domain` nor `create_xx_rule` command, the tools will derive the secondary domain as explained in the following sections.

 *Tip*

In this document, the power domain of a special low power cell instance refers to the primary power domain of that instance.

A derived domain can contain instances that have a secondary power domain definition, such as always-on instances. However the secondary domain of these instances is not always a base domain of the derived domain containing the instances. For the example shown in Figure 4-2 on page 44, `PD4` is not a secondary power domain for `PD3` even though `PD4` is the secondary domain for the always on buffer instance in `PD3`.

Secondary Power Domain of Isolation Instances

If you plan to insert isolation logic in the *from* domain that is being powered down, you may need to use isolation cells with two sets of power supplies. In this case, it is recommended that you specify the secondary power domain for the isolation logic using the `-secondary_domain` option of the `create_isolation_rule` command. The primary power and ground nets of the specified secondary domain will be connected to the secondary power and ground pins of the isolation instances.

The secondary domain of an isolation instance is determined in the following order:

1. The secondary domain specified on the isolation instance using identify_secondary_domain.
2. The secondary domain specified in the corresponding isolation rule for the isolation instance.

3. The power domain of the logic that drives the enable pin.

However, the tools cannot determine the secondary domain of the isolation logic in the following cases:

- The enable expression is a complex expression of individual signals, possibly driven from different power domains or driven from a port without domain assignment (such as the ports in testbench).
- The isolation rule is specified with the `-no_condition` option

If you plan to use single-rail isolation cells, the isolation rule does not need the specification of the secondary domain. The power domain of the isolation instance depends on the location of that instance. If the rule is specified with the `-secondary_domain` option, then the single-rail isolation instance must be instantiated in a domain compatible with the specified secondary domain, which implies that the location domain must be on whenever the secondary domain is on.

The verification tools will detect such cases and issue an error.

During simulation, the isolation logic inferred from the isolation rules must only be considered on if and only if the secondary domain is on. If the secondary domain is shut off, the simulators must consider the output of the inferred isolation logic corrupted.

Secondary Domain of Retention Logic

The primary power domain of a state retention instance is the power domain of its parent module. The secondary power domain of a state retention instance is the domain that controls the retention logic within the cell. The primary power and ground nets of the secondary domain will be connected to the secondary power and ground pins of the state retention instances.

The secondary domain of a state retention instance is determined in the following order:

1. The secondary domain specified on the state retention instance using identify_secondary_domain.
2. The secondary domain specified in the corresponding state retention rule.
3. Use the base domain defined for the primary domain of the retention instance if there is one and only one base domain is specified.

If the secondary domain cannot be determined using the above procedure, it is an error.

The operation mode of the retention logic depends on the state of the primary and secondary domain of the retention logic and which of the two drives the output of the retention logic.

During simulation, the retention logic inferred from the retention rules is considered to retain its state if and only if the secondary domain is on. If the secondary domain is shut off, the simulators must consider the output of the inferred retention logic corrupted.

Secondary Power Domain of Global Cells

It is recommended that you use the identify_secondary_domain command to explicitly set the secondary domain of a global cell. When not specified, the secondary domain of the global cell is assumed to be the power domain of the leaf driver of its data input pin.

For simulation purposes, an instance of a global cell keeps its normal function as long as the secondary domain is on. The simulator will corrupt the output of a global cell instance when the secondary domain is powered down.

Tip

Starting from CPF version 2.0, it is recommended to use global cells instead of always-on cells.

Input and Output Domains of Level Shifters

The definitions of primary and secondary power domains do not always apply to level shifter instances. For level shifters, the more relevant definitions are input domain and output domain, which determine the power/ground connection to the input power/ground pins and output power/ground pins of the cell respectively.

The input power and/or ground pin of the level shifter cell must be connected to the primary power and/or ground net of the input power domain of the level shifter. The output power and/or ground pin of the level shifter cell must be connected to the primary power and/or ground net of the output power domain. For the case of high to low level shifting, the input power domain specification can be ignored.

For a level shifter cell, all pins related to the input power/ground pins belong to the input domain; all pins related to the output power/ground pins belong to the output domain.

Given a level shifter instance in a design, the input domain of the level shifter instance is determined in the following order:

1. Input domain specification in the corresponding level shifter rule
2. Power domain of the leaf driver of this level shifter if the leaf driver is not the output pin of another level shifter cell

3. Power domain of the parent module of the level shifter instance if the valid location of the cell is from
4. Power domain of the parent module of the leaf driver

Given a level shifter instance in a design, the output domain of the level shifter instance is determined in the following order:

1. Output domain specification in the corresponding level shifter rule
2. Power domain of the leaf receiver driven by the level shifter instance if the leaf receiver is not the input pin of another level shifter cell
3. Power domain of the parent module of the level shifter instance if the valid location of the cell is to
4. Power domain of the parent module of the leaf receiver

Power Domains of Pins and Ports

In most cases, the related power domain of an instance pin is the power domain of the instance.

- For more information on how the power domain of an instance is determined, refer to the description of the create_power_domain command.
- Some leaf instances (such as multi-rail isolation cells, level-shifter cells, state retention cells) can also have a secondary power domain defined. For more information, refer to Primary and Secondary Power Domains of Instances on page 46.

For complex macro cells, the domain assignment of the instance itself is not important. Rather, the related power domain of each of its boundary pins is critical to describe the internal power structure of the cell and drive top-level implementations.

The related power domain of a pin or port object is determined as follows:

If a *primary input or primary output port* is part of the boundary port definition of a power domain (using the `-boundary_port` option of `create_power_domain`), the port belongs to that power domain. Otherwise, it belongs to the default power domain of the design.

A *hierarchical pin* does not belong to any power domain unless the corresponding port is a defined as a boundary port, and you specified the `-honor_boundary_port` option in a `set_design` command in a hierarchical CPF file. In this case the power domain of the hierarchical pin belongs to the power domain of the corresponding port.

Single rail standard cell instance pins belong to the power domain of the instance.

In case of *multi-rail standard cell instances*, the power domain assignment of the pins depends on the cell definition as well as on the power domain assignment of the instances (refer to Primary and Secondary Power Domains of Instances):

- For global cells, state retention cells, and isolation cells:
  - the pins related to the switchable power and ground pins belong to the primary power domain of the instances
  - the pins related to the non-switchable power and ground pins belong to the secondary power domain of the instances
- For level shifter cells:
  - the pins related to the input power and ground pins belong to the input domain of the instances
  - the pins related to the output power and ground pins belong to the output domain of the instances

- For more information on the related power and ground pins of the data input and output pins of the special low power cells, such as state retention cell, isolation cell, power switch cell, refer to the description of the `define_xxx_cell` commands in Chapter 9, "define_always_on_cell."

The power domain of a macro cell pin can be directly assigned using the `-boundary_ports` option of the `create_power_domain` command.

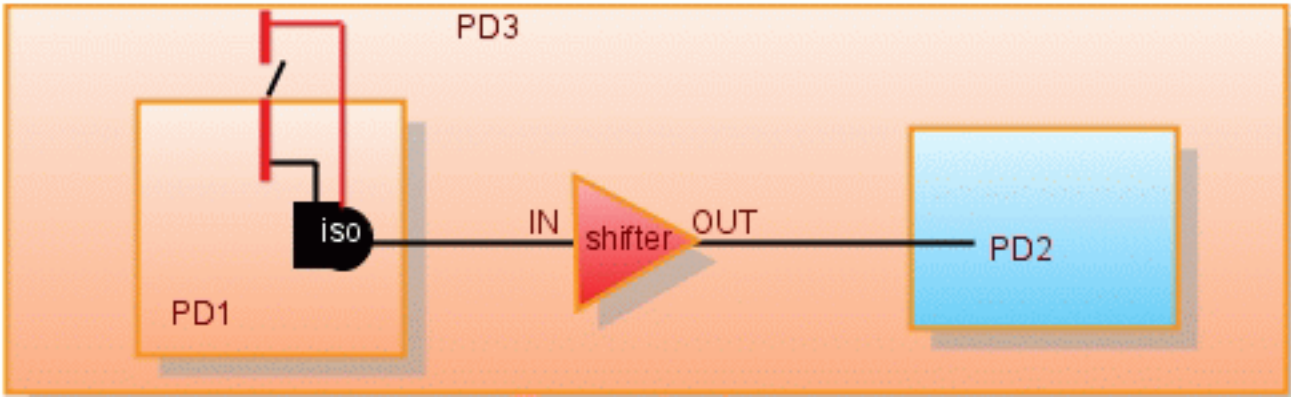
In the hierarchical flow, you can also define the power domain of a macro cell pin using a CPF macro model (see Modeling a Macro Cell for more information). Once the macro cell is instantiated at the chip level, every non-internal switchable power domain of the macro cell must be mapped into a chip-level power domain. The power domain of the macro cell instance pins follow the corresponding cell pin definition or domain mapping specification

Examples

In the following examples, assume that the corresponding level shifter rule has no `-input` or `-output` domain specifications.

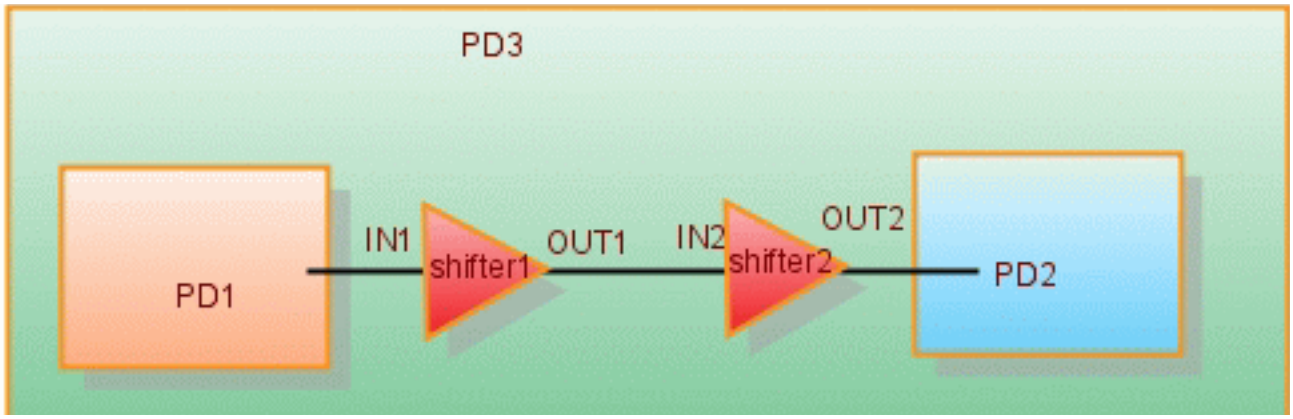
1. In the following example, the level shifter shifts from domain PD1 to PD2, is driven by an isolation cell in domain PD1, but is placed in domain PD3. In addition, the level shifter cell was defined with `-valid_location from`. As a result, PD1 and PD3 must have the same supply voltages in all power mode definitions.

   Input pin IN of the level shifter belongs to PD3, the input domain of the level shifter (because the driver of pin IN—the output pin of isolation cell `ISO`—belongs to PD3, the secondary domain of ISO). The output pin OUT belongs to the output domain of the level shifter which is PD2 (see Input and Output Domains of Level Shifters on page 48).



2. The following example has a back to back level shifter implementation from PD1 (0.8) to PD2 (1.2). The level shifters are located in domain PD3 (1.0). The first level shifter was defined with `-valid_location to`, while the second level shifter was defined with `-valid_location from`.

   According to the semantics (Input and Output Domains of Level Shifters on page 48), pin IN1 belongs to domain PD1, pins OUT1 and IN2 belong to domain PD3, and pinOUT2 belongs to domain PD2.



Modes

Nominal Conditions

A nominal condition is a typical operating condition under which the design or blocks perform. It is defined by the voltages of all power supplies applied to the power domains using this condition. At a minimum, a nominal condition specifies the power supply voltage, but it can also specify the ground supply voltage and PMOS and NMOS bias voltages. Depending on the technology used, this set of voltages determines whether the state of a power domain is on, off or in standby mode.

To define a nominal condition, use the create_nominal_condition command:

```
create_nominal_condition
     -name string
     -voltage {voltage | voltage_list}
    [-ground_voltage {voltage | voltage_list}]
    [-state {on | off | standby}]
    [-pmos_bias_voltage {voltage | voltage_list}]
    [-nmos_bias_voltage {voltage | voltage_list}]
    [-deep_pwell_voltage {voltage | voltage_list}]
    [-deep_nwell_voltage {voltage | voltage_list}]
```

On State

Indicates that all logic in the domain operates at operational speed. To improve performance in the on state, you can apply forward body biasing.

To describe forward body biasing, you must specify -pmos_bias_voltage with a value smaller than the value of -voltage and -nmos_bias_voltage with a value greater than the value of -ground_voltage.

```
create_nominal_condition -name super_fast -voltage 1.2 -ground_voltage 0.0 \
    -state on -pmos_bias_voltage 1.0 -nmos_bias_voltage 0.2
```

Standby State

Indicates that all logic in the domain retain their logic values as long as the inputs are stable. However, if the inputs are changed, the logical values will be corrupted. The standby state can be achieved by reverse body biasing (see example below) or source biasing.

```
create_nominal_condition -name sleep2 -voltage 1.0 -state standby \
    -pmos_bias_voltage 1.2
```

Off State

The logic of a power domain in an `off` state will be corrupted unless the logic is associated with a secondary power domain and the `on` state of this secondary power domain is used to maintain valid values.

```
create_nominal_condition -name power_down -voltage 0.0 -state off
```

Power Modes

A steady state of the design in which some power domains are switched on and some power domains are switched off is called a ***power mode***. In a power mode, each power domain operates at a specific nominal condition.

To define a power mode (or a legal configuration of all power domains in a scope), you need to specify the nominal condition of each power domain in that mode using the create_power_mode command. An unreferenced power domain is considered off.

```
create_power_mode -name drowsy -domain_conditions {PD1@sleep PD2@normal}
```

For each scope, an implicit power mode definition exists where all power domains at that scope operate in the `off` state.

In a hierarchical flow, a power mode can also contain any number of power mode control groups operating at specific power modes. See Power Mode Control Groups for more information.

Generic Modes

A generic mode, also referred to simply as mode, is useful in the early design stage to describe one or more functions of a design. It can be used for early functional verification.

To define a generic mode, use the create_mode command.

A power mode can be considered as a special case of mode. While a power mode requires specification of the states of all the power domains in the current scope, a generic mode does not necessarily associate a specific state with each power domain in the current scope.

For example, if the current scope has 3 domains `PD1`, `PD2`, and `PD3`, and the definition of power mode `foo` omits the condition for `PD3`, it implies is that `PD3` is shut off in mode `foo`.

```
create_power_mode -name foo -domain_conditions { PD1@1v PD2@off }
```

For a generic mode defined the same way, it simply means that the mode applies whenever `PD1` is at 1v nominal condition and `PD2` is shut off, irrespective of the state of `PD3`:

```
create_mode -name newMode -conditions { PD1@1v PD2@off }
```

The example below shows how CPF modes can be used to describe the functionality of a low power design.

```
set_design my_handheld_device
...
create_power_domain -name MAIN -default
create_power_domain -name MULTIMEDIA -instance "audio_mac video_mac"
create_mode -name SLEEP \
    -condition "MAIN@LOWVDD & MULTIMEDIA@OFF & (sleep_reg == 1)" \
    -probability 0.90
create_mode -name ON -condition "MAIN@NOMVDD" -probability 0.10
create_mode -name MM_ACTIVE \
    -condition "@ON & MULTIMEDIA@NOMVDD & (media_active_reg == 1)" \
    -probability 0.03
...
end_design
```

This particular design has two power domains: `MAIN`, which contains the logic for normal operation, and `MULTIMEDIA`, which contains two blocks responsible for audio and video. The `SLEEP` mode describes the state of where `MAIN` is at lower voltage and `MULTIMEDIA` is completely off. The register named `sleep_reg` must be one in order to satisfy this mode. The designers estimate that the design will be in the `SLEEP` mode 90% of the time.

The `ON` mode describes the mode where the `MAIN` domain is operational. The design will be in this mode 10% of the time.

The `MM_ACTIVE` mode is a subset of `ON` mode. In order for the design to be in this mode, the requirements for mode `ON` must be satisfied. In addition, the `MULTIMEDIA` domain must be at the `NOMVDD` condition and the `media_active_reg` must set to one. This mode will be active 3% of the time.

Note: Since there is overlap in modes in the example, the probabilities do not add up to 1.0.

Illegal Domain Configurations

To prevent that an unswitched power domain turns off in relation to other power domains (for example to prevent unisolated internal gates or a high current situation), the assert_illegal_domain_configurations command can be used to explicitly declare a power mode or a configuration of domain conditions as illegal.

At the block-level you define a set of legal configurations of block-level power domains that is crucial for the normal operation of the block. However, domain mapping can cause different configurations of these power domains at the top level which are not legal at the block level.You can use the `assert_illegal_domain_configurations` command to explicitly declare this configuration of domain conditions as illegal at the block level.

To explicitly declare that a particular configuration of domain conditions and/or power mode control group conditions is illegal at the block level, use the `-illegal` option of the create_power_mode command.

If a domain configuration is covered by both a legal and an illegal power mode definition (or through the `assert_illegal_domain_configurations` command), the domain configuration will be considered illegal.

The verification tools should issue a warning message if this happens.

Mode Transitions

A mode transition defines when the design transitions from one power mode to a different power mode. A mode transition can be characterized by specifying

- An integer of number or range of clock *cycles* needed to complete the power mode transition.
- The time (*latency*) needed to complete the power mode transition.

You can also define a start condition and end condition that describe the conditions to trigger and flag the end of the transition.

To define a mode transition, use the create_mode_transition command.

5

Hierarchical Flow

Introduction

Note: *In this context, top-level domain refers to the default power domain of the* ***current*** *design top. In other words, top-level domain does not necessarily refer to the default power domain of the chip.*

IP Categories

In a hierarchical flow, your design can instantiate IPs. CPF distinguishes the following categories:

A ***hard non-custom IP*** is a block that has been synthesized and placed and routed, but small modifications can still be made. It is also referred to as a *hard IP*.

➤ To model a hard IP use the `set_design` command.

A ***hard custom IP*** is a block that is mostly implemented by hand. Custom IP users cannot make any changes to the block and may not have the logical and physical view of the block. Examples are third-party memories. A hard custom IP is also referred to as a *custom IP* or a *macro cell*.

➤ To model a macro cell, use the set_macro_model command.

For more information, see Modeling a Macro Cell.

A ***soft IP*** is a block that is a design module supplied by a third-party. Users have the full capability to (re-)synthesize and place and route the design. Examples are synthesizable CPU cores, and so on.

A soft IP is technology independent. A hard IP (custom or non-custom) is technology dependent.

➤ To model a soft IP use the `set_design` command.

For more information on using an IP in a hierarchical design, refer to CPF Modeling for Hierarchical Design.

Power Domain Mapping Concepts

In a hierarchical design flow, a block can be implemented with a complex power structure. When the block is instantiated at the top level, the power and ground pins of the block must be connected to the power and ground nets of the top-level design. In other cases, designers might want to merge a power domain of a block into another power domain at the top when the block is instantiated at the top.

Power domain mapping is the operation of merging a power domain of a block with another power domain of the top-level design in which the block is instantiated.

Mapping two power domains involves

1. Connecting the primary power and ground pins or nets of the block-level domain to the primary power and ground net of the domain specified at the top level
2. Merging the elements of the power domain of the block into the top-level domain
3. Merging the power modes
4. Resolving the precedence of the power domain settings
5. Resolving the precedence of the top-level and block-level rules

Once a block-level power domain is mapped into a top-level power domain, the two power domains are considered identical and all instances of the two domains share the same power characteristics. For example, the standard cell instances of the two power domains will have their primary power and ground pin (follow pins) connected to the same primary power and ground nets.

Tip

A warning will be given if the default block-leveldomain is mapped into a top-level domain which is different from the domain to which the block instance is assigned at the top level. The block-level domain mapping takes precedence. In the following example, instance `myFoo` is assigned to top-level domain `PD1`. After power domain mapping, `PDX`, the default power domain of the block, is mapped to power domain `PD2` which differs from `PD1`.

```
create_power_domain -name PD1 -instances { i1 myFoo ...} -default
create_power_domain -name PD2 ...
create_power_domain -name PD3 ...

set_instance myFoo -domain_mapping { {PDX PD2} {PDY PD3} }
set_design foo
create_power_domain -name PDX -default -boundary_ports ...
create_power_domain -name PDY -boundary_ports ...
end_design foo
```

The correct way is to not include the macro instance `myFoo` in the top-level domain specification.

```
create_power_domain -name PD1 -instances { i1 ...} -default
create_power_domain -name PD2 ...
create_power_domain -name PD3 ...

set_instance myFoo -domain_mapping { {PDX PD2} {PDY PD3} }
set_design foo
create_power_domain -name PDX -default -boundary_ports ...
create_power_domain -name PDY -boundary_ports ...
end_design foo
```

After domain mapping, macro instance `myFoo` will be in domain `PD2`.

Related commands

set_instance on page 226

set_design on page 212 and end_design on page 195

set_macro_model on page 232 and end_macro_model on page 197

Handling Power Domain Mapping

If the top-level domain is an unswitched domain, all rules defined for the block-level domain need to be reevaluated for their necessity. For example, any state retention rule without `-required` option created for the block-level domain should be ignored by tools when the block is integrated into the top-level design. See Chapter 6, “Precedence and Semantics of the Rules” for more information.

On the other hand, if the top-level domain is a switchable domain, the verification tools should check the following requirements to ensure the correct usage of domain mapping:

- When both the top-level and block-level domains are specified with a shutoff condition, the shutoff conditions must be structurally equivalent.

  Two CPF Boolean expressions are ***structurally equivalent*** if

  a. Each pin in the first expression is electrically connected to a pin or its equivalent control pin in the second expression.

  See also set_equivalent_control_pins on page 218.

  b. The two expressions become identical if each pin in the first expression is replaced with its corresponding pin in the second expression.

- If the block-level domain is an on-chip controlled external switchable domain with base domains, the number of base domains for the block-level domain must correspond to the number of base domains for the top-level domain. In addition, each base domain of the block-level domain must map to a unique base domain of the top-level domain.

An unswitched block-level domain can be mapped into any unswitched top-level domain. However, mapping an unswitched block-level domain into a switchable top-level domain may cause functional failure depending on the actual design of the block. As a result, the verification tools should issue warnings when this occurs. In this case, the block-level domain either inherits the top-level rules completely or just the control conditions. See Chapter 6, “Precedence and Semantics of the Rules” on handling hierarchical rules and isolation and retention rules without conditions for this case.

A block-level internal switchable domain can only be mapped into a top-level domain if the following conditions are met:

- The top-level domain must be either internal switchable or an unswitched virtual power domain.
- Both the top-level and block-level domain must have exactly one base domain specified.

- The base domain of the block-level domain must also be mapped into the base domain of the top-level domain.
- The domain shutoff conditions of the top-level and block-level domain must be structurally equivalent.

Handling Domain Attributes after Domain Mapping

Note: In this section, any options specified with either the create_power_domain or update_power_domain command are referred to as *domain attributes*.

Once a block-level domain is mapped into a top-level domain, the following precedence rules determine how the block-level domain attributes should be evaluated with respect to the settings of the top-level domain:

- `-default_restore_edge/-default_save_edge/-default_restore_level/-default_save_level/-default_isolation_condition`:

  Block-level default conditions always overwrite the top-level default conditions. The top-level default condition will be used if the block-level default condition is missing.

- `-power_down_states/-power_up_states/-transition_slope/-transition_cycles/-transition_latency/-pmos_bias_net/-nmos_bias_net`:

  The settings of the top-level domain precede any settings of the block-level domain. If the top-level domain has no setting, then the default settings will apply. For example, `-power_up_states` applies to all non-retention type registers, flops, or latches of the domain.

- `-boundary_ports`:

  If the CPF model is a macro model or a design specified with the `-honor_boundary_port_domain` option, its boundary ports should be merged into the top-level domain definition. Otherwise, the boundary port association should be ignored.

- `-user_attributes`:

  The settings of both domains should be merged together. The top-level domain should contain the attributes from both power domains.

Handling Power Modes after Domain Mapping

A block-level power mode can be merged into the top-level power modes in one of the following ways:

- All block-level domains are mapped into top-level domains

  In this case, the top-level mode definitions become the current design power modes after the domain mapping.

  Verification tools should flag any inconsistency between the block-level power mode definitions and the top-level power mode definitions. They should give

  - An error message if a power mode at the top level does not match any corresponding modes at the block level before mapping.

    Such a scenario could indicate that an IP is used in a wrong way.

  - A warning if a power mode at the block level does not match any power mode at the top level.

    This can indicate that some configuration of the IP may not be used at the chip level.

  For example, during domain mapping, if a block-level domain is on in all power mode definitions, the corresponding top-level power domain must not be off in any top-level power mode definitions except when all other lower scope domains are off as well.

  Note: For each scope, an implicit power mode definition exists where all power domains at that scope are operating in the off state. For more information, see Modes.

- One or more block-level domains are not mapped into a top-level domain

  These block-level domains become power domains visible at the top-level, and they can be referenced using the hierarchical name for the domains.

  The top-level power mode definition can refer to these block-level domains

  - Directly in the domain condition specification

    For more information on how to name a power domain in a different scope, refer to Referencing CPF Objects.

  - By referencing a block-level mode (that involves these domains) using the power mode control group definitions

    See Power Mode Control Groups for details.

Handling of Initial Statements

Initial statements are non-synthesizable code used in simulation to create proper startup conditions at time zero of the simulation.

To specify the immediate action to be taken when the power is restored to power domains, you can use the set_sim_control command.

You can specify this command in a block-level CPF. It is equivalent to a command at the top level with the `-instances` or `-domain` options to restrict the target selection. For example:

Case 1

Consider the following block level command in block `foo`:

```
set_sim_control -target initial1
```

The top-level equivalent command will be:

```
set_sim_control -target initial1 -instances foo
```

Case 2

Consider the following block level command in block `foo`:

```
set_sim_control -target initial1 -domain X
```

If the block-level domain `X` cannot be mapped to any top-level domain, the top-level equivalent command will be:

```
set_sim_control -target initial1 -domain foo/X
```

Case 3:

Consider the following block level command:

```
set_sim_control -target initial1 -instances {i1 i2 ...}
```

The top-level equivalent command will be:

```
set_sim_control -target initial1 -instances {foo/i1 foo/i2 ...}
```

Modeling a Macro Cell

Proper modeling of macro cells (such as a RAM) with complex power network is important because it serves as a specification for the

- Implementation tool to properly hook up the power and ground pins of the IP at the top level
- Verification tool to check for consistency between the implementation and the constraints specified in the CPF.
- Simulation tool to verify the behavior

When a macro cell has only a non-power-aware behavioral model to describe its functionality, implementation and verification tools have to rely on the CPF modeling of the internal power network using the boundary ports.

When you assign a boundary port of a macro cell to a power domain, it is implied that the logic inside the macro cell connected to this port is powered by the power supply of this domain.

Modeling the Internal Power Structure of a Macro Cell

The design in Figure 5-1 contains a macro cell, `BlockA`, at the SoC level. This section describes how to model the internal power behavior of this macro cell in CPF.

Figure 5-1 Block diagram of Design SoC

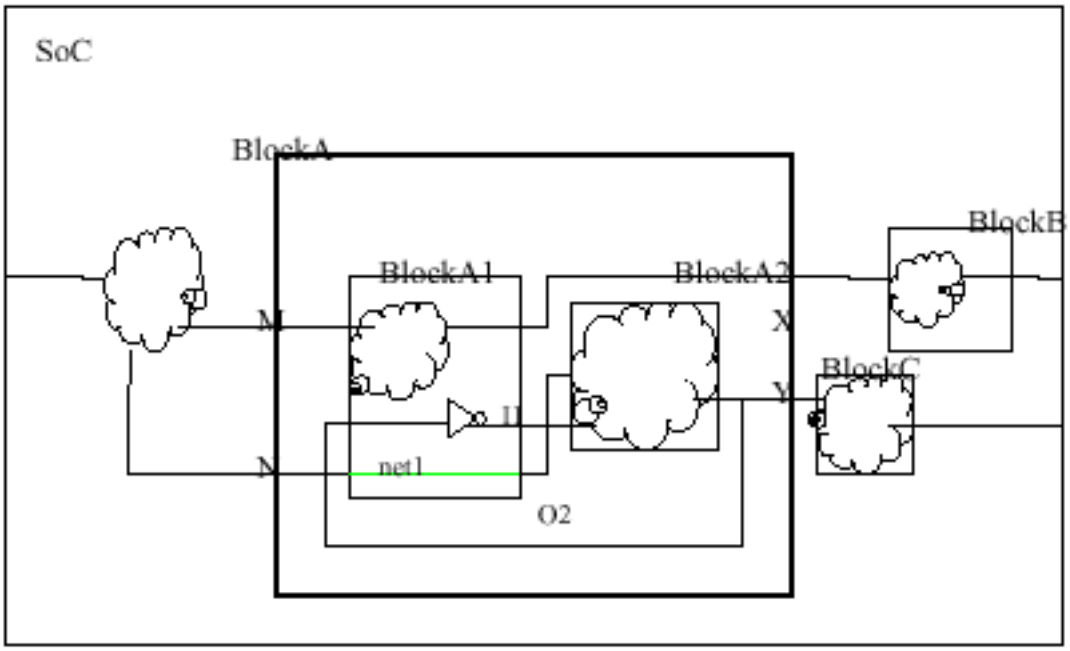
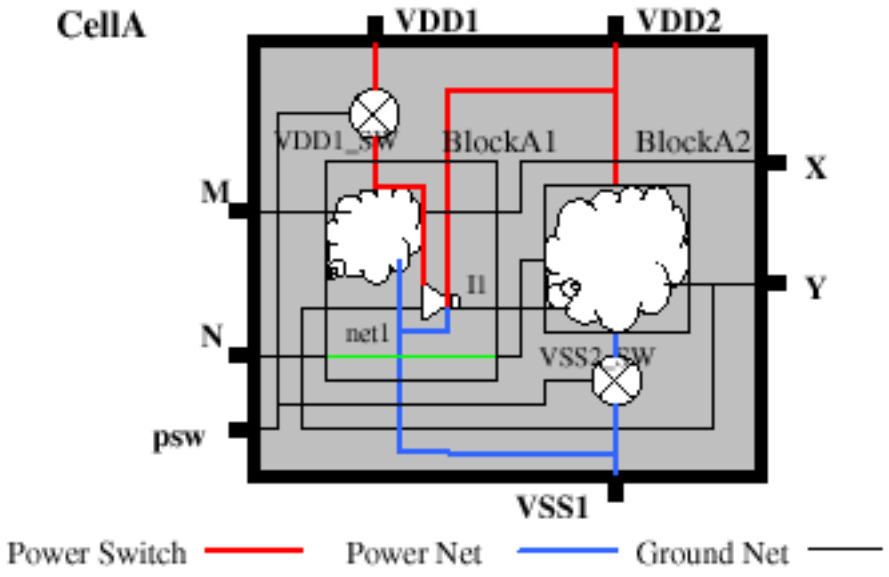


Figure 5-2 on page 67 shows a more detailed view of cell `cellA` of which `BlockA` is an instantiation.

In case of a macro cell, only the (boundary) input and output pins of the cell are visible to the outside world.

In `cellA`, the power shutoff is implemented with power switches that are part of the macro cell. As a result, the control ports for the power switches are part of the boundary port definitions.

Figure 5-2 Macro Cell cellA



To describe the internal power structure of a macro cell, the concept of a CPF *macro model* is introduced. Figure 5-3 on page 68 shows the corresponding CPF to describe this IP cell.

Figure 5-3 CPF file CellA.cpf for Macro Cell CellA

```
set_macro_model CellA

create_power_domain -name PD1 -default -boundary_ports { psw[0]}
create_power_domain -name PD2 -boundary_ports { psw[1]}

create_power_domain -name PD1_SW -boundary_ports {M X}  \
-shutoff_condition { !psw[0] }  -base_domains PD1

create_power_domain  -name PD2_SW -boundary_ports {Y N} \
-shutoff_condition { !psw[1] } -base_domains PD2

update_power_domain -name PD1 -primary_power_net VDD1 -primary_ground_net VSS1
update_power_domain -name PD2 -primary_power_net VDD2 -primary_ground_net VSS1

end_macro_model
```

The definition of a macro model starts with a `set_macro_model` command that specifies the name of the library cell that represents the macro cell. The definition ends with an `end_macro_model` command. These commands delimit the scope of a macro model description.

All commands between `set_macro_model` and `end_macro_model` describe the internal implementation of the macro cell.

Most power domains in the macro model are virtual power domains. The pins specified with the `-boundary_ports` option of the `create_power_domain` command are either input or output pins of the macro cell. The boundary ports belong to the specified domain. If the pin is an input pin, the power domain driven by the pin is the specified domain. If the pin is an output pin, the domain driving the pin is the specified power domain.

The CPF in Figure 5-3 on page 68 contains the following power structure information of the macro cell:

- The macro cell has two unswitched power domains, `PD1` and `PD2`. The corresponding `update_power_domain` command specify the external power and ground ports of the domain using the `-primary_power_net` and `-primary_ground_net` options.
- The macro cell has two internal switchable power domains, `PD1_SW` and `PD2_SW`.

  Even though domain `PD1_SW` is powered down using a power (header) switch, while domain `PD2_SW` can be powered down using a ground (footer) switch, from a modeling point of view, both domains are modeled the same way.

  Both are modeled with an explicit base domain specification. Referring to the `update_power_domain` commands of the base domains, the definition clearly describes the external power and ground ports of power domains `PD1_SW` and `PD2_SW`.
- Output pin `X` of the macro cell is driven by a signal from power domain `PD1_SW`, which is switchable. As a result, proper isolation logic is required at the top-level of the design when the macro cell is instantiated.

- Output pin `Y` of the macro cell is driven by a signal from power domain `PD2_SW`, which is also switchable. As a result, proper isolation logic is required when the macro cell is instantiated.

At the top-level CPF a CPF macro model can be inferred using one of the following methods:

- Use the set_instance command without the `-model` option, and follow the command with the CPF specification of the macro cell as shown in see Figure 5-4 on page 69.
- First load the CPF specification of the macro cell, then use the `set_instance` command with the `-model` option to reference the CPF macro model as shown in Figure 5-5 on page 69.

The specified instance must be an instantiation of the cell specified by `set_macro_model`.

Figure 5-4 Chip-Level CPF inferring macro cell CellA (method 1)

```
set_design SoC
create_power_domain -name PDA -default
create_power_domain -name PDB -instances BlockB
create_power_domain -name PDB_SW -instances BlockC -shutoff_condition { !psw3}
create_isolation_rule -name iso1 -from PDB_SW  -isolation_enable iso_en
set_instance BlockA -domain_mapping {{PD1 PDA} {PD2 PDB}}
include CellA.cpf
end_design
```

Figure 5-5 Chip-Level CPF inferring macro cell CellA (method 2)

```
include CellA.cpf
set_design SoC
create_power_domain -name PDA -default
create_power_domain -name PDB -instances BlockB
create_power_domain -name PDB_SW  -instances BlockC -shutoff_condition { !psw3}
create_isolation_rule -name iso1 -from PDB_SW  -isolation_enable iso_en
set_instance BlockA -model CellA -domain_mapping {{PD1 PDA} {PD2 PDB}}
end_design
```

In the scenario shown in Figure 5-6 on page 70, the macro cell has some internal logic to control the power domains inside the macro cell. As a result, some required interface logic such as isolation logic may already be inserted in the macro cell. For example, output port `X` is already properly isolated in the macro cell. Consequently, the CPF in Figure 5-3 on page 68 needs to be modified slightly to reflect the fact that output pin X is already isolated. The CPF for the macro cell in Figure 5-6 is shown in Figure 5-7 on page 70.

Figure 5-6 Macro Cell in Figure 5-2 where all external ports are properly isolated

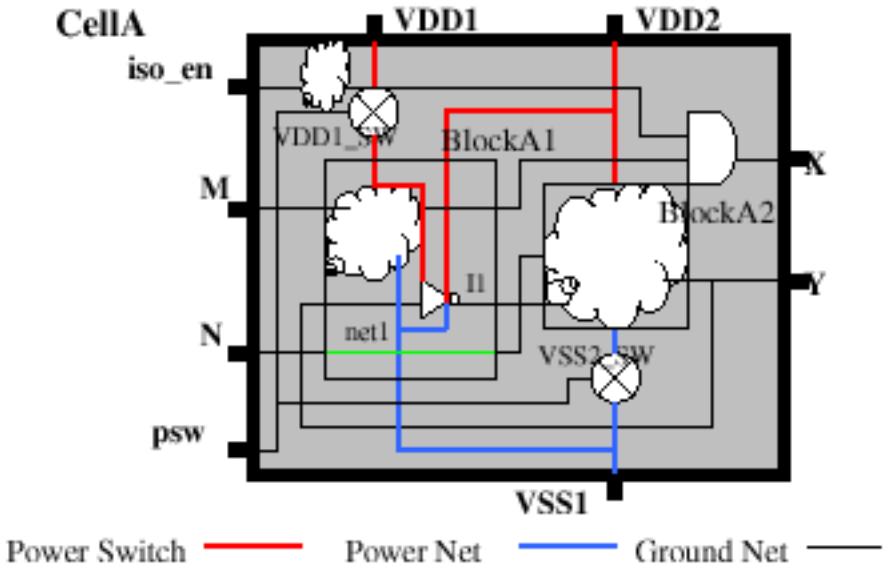


Figure 5-7 CPF for the Macro Cell in Figure 5-6 with output pins already isolated within the Macro Cell

```
set_macro_model CellA

create_power_domain -name PD1 -default -boundary_ports { X {psw[0]} iso_en}
create_power_domain -name PD2 -boundary_ports { psw[1]}

create_power_domain -name PD1_SW -boundary_ports {M}  \
-shutoff_condition { !psw[0] }  -base_domains PD1

create_power_domain  -name PD2_SW -boundary_ports {Y N} \
-shutoff_condition { !psw[1] } -base_domains PD2

update_power_domain -name PD1 -primary_power_net VDD1 -primary_ground_net VSS1
update_power_domain -name PD2 -primary_power_net VDD2 -primary_ground_net VSS1

create_isolation_rule -name iso -pins X -from PD1_SW \
-isolation_condition iso_en -isolation_output low

end_macro_model
```

Comparing Figure 5-7 with 5-6, the new CPF describes that pin `X` now belong to the unswitched domain `PD1` because pin X is already isolated to low when `iso_en` is `1`. Note that all commands between `set_macro_model` and `end_macro_model` are only for documenting purpose since they describe a macro cell. The `create_isolation_rule` command specifies that pin X of the macro cell is *already* isolated. After applying the same chip level CPF (see Figure 5-4), all implementation information is there. There is no need for isolation from pin X to `BlockB` since both domains that drive X and the domain of `BlockB` are unswitched.

Note: A complete isolation rule in a macro cell describes the isolation logic implemented inside the macro cell. Consequently, this isolation logic must be modeled explicitly in the behavioral model of the macro cell. An incomplete isolation rule (a rule with neither `-isolation_condition` nor `-no_condition`) specified for the input of a macro model indicates the required isolation value when the power domain that drives the port is powered down.

Modeling the Internal Power Behavior of a Macro Cell

The behavioral simulation model of a macro cell may not have any logical hierarchy. In this case, the methodology to create power domains following logical hierarchies does not work. To enable verification of such macro cells, the IP designer must use the following approach:

1. Within the scope of a macro model definition, create a power domain with registers only.
2. Create a default domain that is unswitched.
3. Create all necessary domains to model the switchable power domains and only list registers in the behavioral model as its members.
4. Associate the primary inputs and outputs of the IP with the proper power domains depending on the logic it connects to or is driven by.

   Note: The association of an input or output port with a power domain depends on the logic connected to the ports. For example, if the IP has primary output ports driven by logic that will be powered down, the ports should be assigned to that power domain and the outputs will be asserted X when the domain is down. However, if the port is isolated, the port should be assigned to the secondary power domain of the isolation instance.

To support this modeling technique, the simulation tools must inject an X and the emulation tools should use the value specified for the `-power_down_states` option at the following ports when a power domain is powered down:

- Associated primary input ports
- Associated output ports that have no isolation rules associated with them
- Outputs of registers in the domain that are not part of any retention rules

Important

The power down behavior of non-register logic is simulated by injecting X at primary inputs and unisolated outputs. Injecting X at unisolated outputs is needed because X may not be able to propagate if there is a controlling signal at some logic. For example, if `y = a|b`, `y` will not be `x` when `a` is `1` and `b` is `x`. The power down behavior of registers is simulated directly by associating them with the proper power domains.

Exceptions

The above approach will not work when a primary input port fans out to multiple power domains with different switching behavior. In this case, the input port should be treated as always on during simulation. However, to accurately model such a macro cell, it is recommended to make their behavioral model CPF compliant by creating a module hierarchy based on the power domain partitions.

Example

Using the macro cell modeling technique, CPF can model complex multi-rail RAM. Lets consider a RAM macro cell with three power rails. One power rail `VDD_MAIN` is used for peripheral logic such as address encoder/decoder. The second power rail `VDD_ARRAY` is used for the memory array. The third power rail `VDD_AO` is used for the rest of the logic and it is always on. The peripheral logic has an internal power switch which can shut off the power supply to the peripheral logic when `CE` is low. The power net for the array is external switchable. The following CPF describes the RAM model.

Figure 5-8 RAM Model with Internal Power Switch

```
set_macro_model RAM
create_power_domain -name PD1 -boundary_ports {...} -shutoff_condition !CE \
-base_domains PD3
create_power_switch_rule -name sw1 -domain PD1 -external_power_net VDD_AO
create_power_domain -name PD2 -boundary_ports {...} -instances mem* \
create_power_domain -name PD3 -boundary_ports {...} -default
update_power_domain -name PD1 -primary_power_net VDD_MAIN \
-primary_ground_net VSS
update_power_domain -name PD2 -primary_power_net VDD_ARRAY \
-primary_ground_net VSS
update_power_domain -name PD3 -primary_power_net VDD_AO \
-primary_ground_net VSS
create_nominal_condition -name on -voltage 1.0 -state on
create_nominal_condition -name off -voltage 0 -state off
create_power_mode -name all_on -domain_conditions { PD1@on PD2@on PD3@on} -default
create_power_mode -name drowsy -domain_conditions { PD1@on PD2@off PD3@on}
create_power_mode -name sleep -domain_conditions { PD1@off PD2@off PD3@on}
end_macro_model
```

`mem*` are the registers in the memory array in the simulation model. Power domain `PD2` is an unswitched domain at the block level, but the macro model specification (power mode `drowsy` and `sleep`) indicates that it can be switched off externally. `PD2` is mapped into a top-level switchable domain when the macro cell is instantiated as shown in Figure 5-9.

Figure 5-9 Top Level Instantiation of RAM Can Map Domains in RAM to Any Top-Level Domain

```
include RAM.cpf

set_design top

create_power_domain -name PD_VDDA -default
create_power_domain -name PD_VDDB -instances {BlockB}
create_power_domain -name PD_VDDB_SW -instances {BlockC} \
-shutoff_condition { !psw3}

set_instance iRAM -model RAM -domain_mapping {{PD2 PD_VDDB_SW} {PD3 PD_VDDA}}

end_design
```

Note: Above CPF can also be used for the testbench module so the array power down behavior can be simulated and verified.

CPF Modeling for Hierarchical Design

In a hierarchical design flow, a block (soft IP, hard IP or custom IP) can have a separate CPF file to describe its power intent. The block level CPF is referred to as ***CPF model*** for the block. A CPF model is either a power design model or a macro model.

In CPF, a CPF model can be instantiated in a design in one of the following ways:

- First use the include command to load the CPF specification, then use the set_instance command with the
  - `-model` option to reference a CPF model for a macro cell instance (custom IP)
  - `-design` option to reference a CPF power design model for a module (hard or soft IP instance) (see Example 5-1 on page 75)

  A `set_instance` command specified with an instance name with a `-design` or `-model` option will not cause a scope change after the command is executed.
- Use the `set_instance` command without either of the above options, and follow the command by either the set_design or set_macro_model command. Some commands are allowed between `set_instance` and `set_design` or `set_macro_model`. See the commands shown in the Command Dependency table before `set_design`.

  In this case (see Example 5-2 on page 76) the following steps will occur:

  a. The scope will be changed to the specified instance.

  b. The commands between the next `set_design` and `end_design` pair or `set_macro_model` and `end_macro_model` pair will be loaded and applied to the specified instance.

  c. The scope will be reset to the scope prior to the `set_instance` command after the `end_design` or `end_macro_model` command.

The same CPF model can be bound to multiple instances using `set_instance` with the `-design` or `-model` options. By default, a CPF model can be bound to any instance of any module (if defined with `set_design`) or macro cell (if defined with `set_macro_model`). If the CPF model definition specifies module (using `set_design -modules`) or cell names (using `set_macro_model -cells`), then only instantiations of modules or macro cells whose names match those in the definitions can be bound.

Power design definitions are scope sensitive.

- It is legal to have CPF models created with the same name in different scopes.

- If multiple power designs are created with the same name in the same scope, the first definition will be used and all other definitions will be ignored (see Example 5-4 on page 76 and Example 5-8 on page 78).

  Note: In CPF 1.1, the power intent was appended using additional `set_design` commands with the same name in the same scope. In CPF 2.0, this is accomplished using the update_design command.

Macro model definitions are **not** scope sensitive.

- If multiple macro cell models are created with the same name, the first definition will be used and all other definitions will be ignored (see Example 5-3 on page 76).

The power intent specified by update_design shall be treated as part of the original power design specification and be applied to all set_instance commands that use this power design model, including the set_instance commands that precede update_design (see Example 5-7 on page 77).

Tip

To prevent warnings in environments where the same CPF file might get included multiple times, coding techniques such as the use of Tcl `if` constructs and environment variables can be used to prevent duplicate CPF model definitions.

In the following examples, assume `foo` is a power design for a Verilog module instantiated as `I1`, `I2`, and `I3` in RTL. After synthesis uniquifies the module, the module name for I1, I2 and I3 may no longer be `foo`.

Example 5-1

```
set_design foo ;# model 1 for foo
...
end_design
set_instance I1 -design foo -domain_mapping ...
set_instance I2 -design foo -domain_mapping ...
```

`I1` and `I2` will be bound to the power design model foo.

Example 5-2

```
set_design foo ;# model 1 for foo
...
end_design
set_instance I1 -design foo -domain_mapping ...
set_instance I2 -domain_mapping ...
set_design foo ;# model 2 for foo
...
end_design
set_instance I3 -design foo -domain_mapping ...
```

`I1` and `I3` will be bound to the first model of `foo`, while `I2` will be bound to the second model. In this case, the second definition of `foo` belongs to the scope of I2.

Example 5-3

```
set_macro_model foo ;# model 1 for foo
...
end_macro_model
set_macro_model foo ;# model 2 for foo
...
end_macro_model
set_instance I1 -model foo -domain_mapping ...
```

`I1` will be bound to the first CPF macro model for foo. The second macro model is ignored and a warning will be issued.

Example 5-4

```
set_design foo ;# first definition of model foo
create_power_domain -name PD1 -default
...
end_design
set_design foo ;# second definition of model foo
update_power_domain -name PD1 -primary_power_net VDD1
...
end_design
set_instance I1 -design foo -domain_mapping ...
```

`I1` is linked to the first model for `foo`. The second model for `foo` is ignored and a warning is issued.

Note: In CPF1.1, when the update_design command did not exist, the second model of `foo` would have been appended to the first.

Example 5-5

```
set_design foo ;# first definition of model foo
create_power_domain -name PD1 -default
...
end_design
update_design foo ;# append to definition of model foo
update_power_domain -name PD1 -primary_power_net VDD1
...
end_design
set_instance I1 -design foo -domain_mapping ...
```

In this example, the content of the second definition of model foo is appended to the content of the first definition of model foo and instance `I1` is bound to the resulting model. The following is the equivalent CPF:

```
set_design foo ;
create_power_domain -name PD1 -default
...
update_power_domain -name PD1 -primary_power_net VDD1
...
end_design
set_instance I1 -design foo -domain_mapping ...
```

Example 5-6

```
set_instance I1 -domain_mapping ...
set_design foo ;# model 1 for foo
...
end_design
set_instance I2 -design foo -domain_mapping ...
set_design foo ;# model 2 for foo
...
end_design
```

This CPF generates an error because CPF model `foo` does not exist in the scope where and when `I2` is instantiated. The first model of `foo` belongs to the scope of `I1` and is not accessible outside of that scope. The second model of `foo` is defined after the instantiation of `I2`.

Example 5-7

```
set_design top
set_design foo
...
end_design foo
set_instance I1 -design foo -domain_mapping ...
...
update_design foo
...
end_design foo
...
set_instance I2 -design foo -domain_mapping ...
...
end_design top
```

The original power design foo is appended with power intent defined in the `update_design` command. Both `I1` and `I2` are bound to the fully updated definition of power design `foo`.

Example 5-8

```
set_design top
set_design foo ;# first definition of model foo
...
end_design foo
set design nested_design
...
set_design foo ;# second definition of model foo
...
end_design foo
...
end_design nested_design
set_instance I1 -design foo -domain_mapping ...
...
end_design top
```

The second definition of `foo` is ignored and a warning is issued because two power designs with the same name `foo` were defined in the top scope. (Only `set_instance` causes a scope change.) Instance `I1` refers to the first definition of power design `foo`.

Example 5-9

```
set_design top
set_design foo ;# first definition of model foo
...
end_design foo
set_instance N1 -domain_mapping ...
set design nested_design
...
set_design foo ;# second definition of model foo
...
end_design foo
...
end_design nested_design
set_instance I1 -design foo -domain_mapping ...
...
end_design top
```

This example is similar to Example 5-8 on page 78, but has a `set_instance` command inserted before the `set design nested_design` command. The `set_instance` command causes a scope change and as a result the two definitions of power design `foo` are valid. The first definition belongs to the scope `top`, while the second definition belongs to the scope `N1`. I1 will be bound to the first definition.

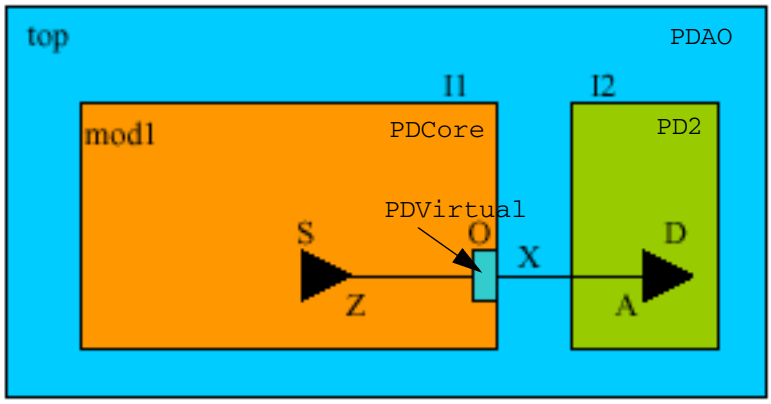
Handling Boundary Port Domain Definition at Top Level

In a hiearchical design flow, each boundary port of an IP block will be assigned to a power domain. This implies that the logic that is outside the IP block and that is connected to this port is powered by the power supply of this domain. This will enable the tools to verify and implement the IP block using the block-level CPF. When the IP block is instantiated at the top level, the block-level boundary port becomes a hierarchical pin at the top level. By default, the hierarchical pin is not associated with any power domain. In a bottom-up hierarchical design flow, designers sometimes want to treat the block-level CPF as a golden specification. The power domain association with the boundary port (at block level) is part of this golden specification. In other words, designers expect that after the top-level implementation the hiearchical pin connection should be consistent with the block-level boundary port domain association.

To support this expectation, the `-honor_boundary_port_domain` option is added to the set_design command. The option has the following implications:

1. This option only applies to the primary input and output ports of the current design (specified by the `set_design` command)
2. If this option is specified, each association of a boundary port with a power domain becomes a design constraint at the top level after the block is instantiated at the top. At the top level, each corresponding hierarchical pin becomes a virtual leaf-level driver or leaf-level load for power-domain traversal purposes where the power domain is the power domain definition for the boundary port.
3. If this option is not specified, the boundary port domain definition does not treat the hierarchical pin as a leaf-level driver or leaf-level load. The tools should traverse through the hierarchical pin when traversing the net connected to the pin till they find the leaf-level driver or leaf-level load.
4. For custom models, each boundary port definition should automatically be treated as a design constraint.

In the following example, module `mod1` is an IP block with a default switchable domain. Also, the output port `O` is declared as part of a virtual domain which is unswitched.




```
set_design mod1 -honor_boundary_port_domain
create_power_domain -name PDVirtual -boundary_ports O
create_power_domain -name PDCore -default -shutoff_condition en \
    -base_domains PDVirtual
end_design
set_design top
create_power_domain -name PDAO default
create_power_domain -name PD2 -shutoff_condition en2 -instances I2
set_instance I1 -design mod1 -domain_mapping {{PDVirtual PDAO}}
end_design
```

When module `mod1` is instantiated at the top, block-level domain `PDVirtual` is mapped to top-level domain `PDAO`. Since block `mod1` is specified with the `-honor_boundary_port_domain` option, then at top level, even though the hierarchical pin `I1/O` has leaf-level driver `I1/S/Z` in domain `I1/PDCore` and leaf-level load `I2/D/A` in domain `PD2`, the boundary domain association of `O` with `PDAO` (due to the domain mapping) should be treated as a design constraint. As a result, the implementation tool may insert a buffer for the top net x within power domain `PDAO` to meet the design requirement.

Power Mode Control Groups

A power mode control group is a set of power domains with an associated set of power modes and mode transitions that apply to this group only. All power modes defined for this group can only reference power domains within this group. All power mode transitions can only reference power modes defined for this group.

A power domain can belong to more than one power mode control group. However, each power mode control group must have one and only one default power mode.

A power mode group can contain other power mode groups. If a power mode control group has another power mode control group as its member and the other power mode control group is not referenced in any power mode definition, the other power mode control group is assumed to be in the default power mode of that group.

The `create_analysis_view` command is also extended to be able to create a top-level analysis view by using block-level analysis views created within a power mode control group. This enables the reuse of the block level CPF analysis view. For example, the operating corner assignment for the block-level power domains can be reused at the top level in the implementation flow.

If a power mode control group in a lower scope is not referred to by another power mode or analysis view in an upper scope, the power mode or analysis view at the block level will be ignored at the top level. However, if a power mode control group in a lower scope is referred by some other power mode or analysis view in an upper scope, but not by all, the power mode control group is supposed to be in the default mode in those modes in the upper scope where it is not referenced.

Default Power Mode Control Group

A *default* power mode control group is automatically created for each CPF scope. The default power mode control group has no name.To reference the default power mode control group of a scope, use the hierarchical path from the current scope to the targeted scope.

The default power mode control group for a scope contains all those power domains defined in this scope that are not declared as members of an explicitly defined power mode control group. In other words, if all power domains of a scope belong to an explicitly defined power mode control group, the default power mode control group has no members.

User-Created Power Mode Control Group

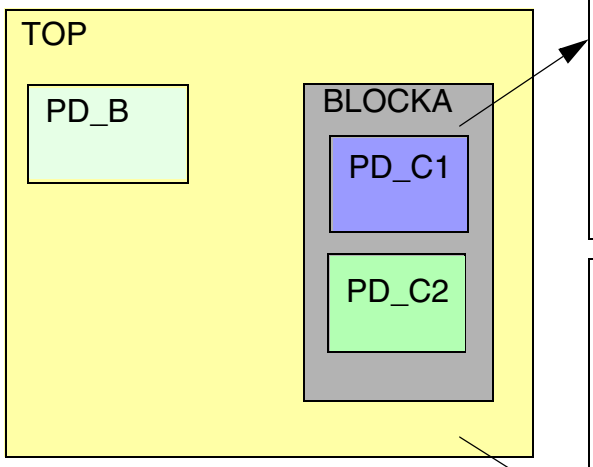
The set_power_mode_control_group and end_power_mode_control_group commands define the start and end of an explicit power mode control group definition.

To declare another power mode control group as a group member of a power mode control group, use the `-groups` option of the `set_power_mode_control_group` command.

Within a scope, you can create one or more power mode control groups. A power mode control group can be referenced by its hierarchical name in top-level CPF commands.

A power domain that is not part of any power mode definition within a power mode control group is assumed to be powered down.

Example 5-10 Using Default Power Control Group




```
#BLOCKA.cpf
set_design BLOCKA
create_power_domain -name PD_C1
create_power_domain -name PD_C2

create_power_mode -name M1 -default \
  -domain_conditions {PD_C1@off PD_C2@on}
create_power_mode -name M2 \
  -domain_conditions {PD_C1@on PD_C2@on}
create_power_mode -name M3 \
  -domain_conditions {PD_C1@on}

create_mode_transition -name CT1 -from M1 \
  -to M2
create_mode_transition -name CT3 -from M2 \
  -to M1
...
end_design
```

```
set_design TOP
create_power_domain  -name PD_B
set_instance INST_A
include BLOCKA.cpf
set_power_mode_control_group -name top \
  -domains PD_B -groups INST_A
create_power_mode -name T1  \
  -domain_conditions { PD_B@on}
create_power_mode -name T2  \
  -domain_conditions { PD_B@off }  \
  -group_modes {INST_A@M2}
create_power_mode T3 -default  \
  -domain_conditions { PD_B@on}
  -group_modes {INST_A@M3}
create_mode_transition -name TT1 -from T1 \
  -to T2
create_mode_transition -name TT2 -from T2 \
  -to T3
create_mode_transition -name TT3 -from T3 \
  -to T1
end_power_mode_control_group
end_design
```

BLOCKA in Example 5-10 is a hierarchical block. The CPF file at the top right defines the power domains, power modes, and power mode transitions for this block. The power modes and power mode transitions are defined with respect to the power domains at this level of hierarchy in the design.

Since no explicit power mode control group is created for BLOCKA, the default power mode control group contains the two power domains of BLOCKA. In this group, power domain PD_C2 is not referenced in mode M3. Therefore, domain PD_C2 is assumed to be in the off state. When the block is instantiated, the hierarchical scope name of the block must be used to refer to the default power mode control group, INST_A in this case.

The CPF file for design TOP has its own power mode control group (top) which includes power domain PD_B and power mode control group INST_A. When defining a power mode for design TOP, you can use the -domain_conditions option for any power domains at this level of hierarchy, but to refer to the power mode of BLOCKA you must use the -group_modes option:

```
create_power_mode -name T2 -domain_conditions {PD_B@off} -group_modes {INST_A@M2}
```

In this case, top-level mode T2 is created with domain PD_B at nominal condition off and power mode control group of BLOCKA (INST_A) in power mode M2, in which both block-level domains PD_C1 and PD_C2 are on.

Top-level mode T1 is created with domain PD_B at nominal condition on. Since the power mode control group (INST_A) is not referenced in this top-level mode, it is assumed to be in the default mode of the power mode control group, which is mode M1.

Example 5-11 Use of Explicitly Created Power Mode Control Groups

```
# from file BLOCKQ.cpf

set_design BLOCKQ
create_power_domain -name PDA1
create_power_domain -name PDA2
create_power_domain -name PDB1
create_power_domain -name PDB2

# PDA1 and PDA2 belong in PMCGA
set_power_mode_control_group -name PMCGA -domains {PDA1 PDA2}
create_power_mode -name MA1 -domain_conditions {PDA1@on PDA2@on } -default
create_power_mode -name MA2 -domain_conditions {PDA1@on PDA2@off}
create_power_mode -name MA3 -domain_conditions {PDA1@off PDA2@on }
create_power_mode -name MA4 -domain_conditions {PDA1@off PDA2@off}
end_power_mode_control_group

# PDB1 and PDB2 belong in PMCGB
set_power_mode_control_group -name PMCGB -domains {PDB1 PDB2}
create_power_mode -name MB1 -domain_conditions {PDB1@on PDB2@on } -default
create_power_mode -name MB2 -domain_conditions {PDB1@on PDB2@off}
create_power_mode -name MB3 -domain_conditions {PDB1@off PDB2@on }
create_power_mode -name MB4 -domain_conditions {PDB1@off PDB2@off}
end_power_mode_control_group

end_design
```

The power mode control group allows you to concisely specify power modes. In this example, the result of creating two power mode control groups with four power modes each is equivalent to explicitly creating 16 power modes.

Example 5-12 Explicit Power Mode Control Group Hierarchical References

```
set_design TOP

create_power_domain -name PD1

set_instance INST_Q
include BLOCKQ.cpf

# assumes INST_Q/PMCGA and INST_Q/PMCGB are in the respective default
# power modes (MA1 and MB1)
create_power_mode -name T1 -domain_conditions {PD1@on} -default

# assumes INST_Q/PMCGB is in its default power mode MB1

create_power_mode -name T2 -domain_conditions {PD1@off} \
-group_modes {INST_Q/PMCGA@MA2}

# assumes INST_A/PMCGA is in its default power mode MA1

create_power_mode -name T3 -domain_conditions {PD1@on} \
-group_modes {INST_Q/PMCGB@MB3}

end_design
```

This example uses the instantiation of `BLOCKQ` from Example 5-11 and illustrates references to the power mode control groups within `BLOCKQ`.

Example 5-13 User-Created Power Control Groups with Domains in Multiple Groups

Assume a design with power domains, CPF, GPU, and MEM. The power modes specify a relationship between two out of the three domains: CPF and GPU, CPU and MEM.

To describe the above intent, you can create two power mode control groups, one containing domains CPU and GPU, and one containing domains CPU and MEM. Note that domain CPU belongs to two groups.

```
set_power_mode_control_group -name PU -groups {CPU GPU}
    create_power_mode -name PU_1 -group_modes {CPU@C1 GPU@C2 }
    create_power_mode -name PU_2 -group_modes {CPU@C3 GPU@C1 }
end_power_mode_control_group
set_power_mode_control_group -name CMEM -groups {CPU MEM}
    create_power_mode -name CMEM_1 -group_modes {CPU@C1 MEM@C2 }
    create_power_mode -name CMEM_2 -group_modes {CPU@C3 MEM@C1 }
end_power_mode_control_group
```

Now, if the current state has CPU@C3, GPU@C1, and MEM@C1, it can be determined that for group PU, the mode is PU_2 and for group CMEM the mode is CMEM_2. Each power mode control group can define the mode transitions and check if the transition is legal. A transition in one group can cause an illegal transition to be reported in another group. For instance, if PU group has a PU_1 to PU_2 transition, and CMEM group does not have a CMEM_1 to CMEM_2 transition, then the CPU domain would go from condition C1 to C3 and an error would be flagged for the CMEM group.

6

Precedence and Semantics of the Rules

Different Categories of Rules

The CPF language uses the concept of *rules* to describe the low power intent of the design—that is, describe the type of low power logic that needs to be added to the design. The CPF language supports the following types of rules:

- Level shifter rules
- Isolation rules
- State retention rules
- Power switch rules

A CPF rule can be associated with a design object either explicitly or implicitly. There are two categories of CPF rules.

- **Specific** rules *explicitly* specify the targeted design objects. For example,
  - Isolation or level shifter rules specify the targeted design objects with the `-pins` option or `-exclude` option, or both options.
  - State retention rules specify the targeted design objects with the `-instances` option or `-exclude` option, or both options.

  Note: Wildcards are considered to specify the targeted design objects explicitly (by expanding the string to a list).
- **Generic** rules do not explicitly specify the targeted design objects, but the targeted design objects can be *derived* from some option specified with the rule. For example,
  - Isolation or level shifter rules specified with `-from` or `-to` options target domain crossings driven by logic in the domains specified with the `-from` option or driving logic in the power domains specified with the `-to` option.
  - State retention rules specified with the `-domain` option target all registers or sequential instances in the specified power domain.

Tip

For isolation and level shifter rules, the targeted design objects refer to hierarchical pins or domain crossings. For state retention rules, the targeted design objects refer to registers or sequential instances.

Note: A domain crossing is a connection between one driver and one load in two different power domains.

Note: A power switch rule is always associated with a current design. Therefore each power switch rule can be considered a specific rule.

Rules in Presence of Existing Power Logic

A CPF file is normally written for an RTL description to specify the low power intent. (CPF rules specify where the user wants to *add* power logic in the design.)

This CPF may be used as a golden specification at a lower abstract level for verification or implementation. However, the HDL description may already contain some low power logic independent of what is specified in the CPF file. A netlist can contain low power logic that is either

- independent of what is specified in the CPF file
- generated based on what is specified in the CPF file

If the design objects targeted by a CPF rule already have some low power logic, the logic in the HDL description takes precedence and the CPF rule is ignored. In other words, the CPF rules specified with the `create_xx_rule` commands will not be used to generate new low power logic in the design if the targeted design objects already have the required low power logic associated with them. Tools that check for the consistency between CPF and the netlist should report when the existing logic does not match the requirements from the CPF file.

Identifying Existing Power Logic

The CPF language also provides support to *identify existing* power logic in the HDL description or in the given netlist.

It is possible to

- Identify instances that are instantiations of library cells defined by CPF `define_xxx_cell` commands.
- Identify non-dedicated or generic isolation logic specified in the `identify_power_logic` command.

Even if you instantiated the isolation logic in the RTL description, you must still define the isolation rules to describe your intention for the isolation logic if you want to use the CPF-based flow.

Precedence of Rules in the Flat Flow

If several rules of the same type apply to the same design objects (see Different Categories of Rules, for more information on the design objects), the following order of precedence applies (arranged from highest to lowest priority):

1. Specific rules specified with the `-force` option
2. Specific rules specified with both the `-from` and `-to` options
3. Specific rules specified with only `-from` or `-to` option
4. Generic rules specified with both the `-from` and `-to` options
5. Generic rules specified with only `-from` or `-to` option

If multiple CPF rules with the same priority are applied to the same design objects

- It is an error if all rules are specified with -force.
- Otherwise, the last rule takes precedence. In this case, the tools will issue a warning message.

Note: It is possible that the rule with the highest priority cannot be implemented. (For example, the required cell is not available.) In this case, the tools will give a warning message.

Note: For isolation rules and level shifter rules, the `-location` option does not influence the precedence.

The precedence rules apply independently for each value for the `-isolation_target` option.

Precedence of Rules in the Hierarchical Flow

In the hierarchical flow, you can have a CPF file for a module at the upper level of the hierarchy and a CPF file for a module at a lower level in the hierarchy. In the following, the upper-level module is referred to as the top-level design and the lower-level hierarchy is referred to as the block-level design.

The following policies are followed by all tools when a block-level CPF is integrated into the top-level CPF using the `set_instance` command:

1. Block-level rules always overwrite the top-level rules if both rules apply to the same objects at the block-level after domain mapping.
2. Top-level rules are only applied to block-level objects if the block-level objects have no rule specified and the block is not a macro model.

   In this case, the tools should issue a warning to inform users that top-level CPF rules are applied to the block.
3. Block-level state retention, isolation, and level shifter rules do not apply to objects outside the scope of the block. Block-level rules are promoted to the top level, but they only apply to objects in the block.

Note: For more information on how multiple rules on the same signal or net are handled, refer to Multiple Level Shifter and Isolation Rules.

Rules Semantics

- Level Shifter Rules
- Isolation Rules
- Multiple Level Shifter and Isolation Rules
- State Retention Rules

Note: *In this context,* ***net*** *refers to logical net segment.*

Level Shifter Rules

A level shifter rule specifies one or more domain crossings which require level-shifter logic. Level-shifter logic can be required because the leaf drivers and leaf loads of the net have different supply voltages, or because it is part of the design intent.

Note: An exception can be made for some special pins for which a user-specified input voltage tolerance is defined (see `define_power_switch_cell` and `set_input_voltage_tolerance` commands).

The targeted nets are selected by the options `-from`, `-to`, and `-pins` of the create_level_shifter_rule command.

This section contains the following topics:

Interpretation of Different Options

- `-force` specifies to apply the rule to the specified pin without any of the filtering based on driver and load analysis.

  Note: The `-force` option must be specified with the `-pins` option. The `-from` or `-to` option will be ignored if they are combined with the `-force` option.

- `-from` specifies to apply the rule to those nets driven by logic in the specified *from* domains and are driving logic in other power domains.

- `-to` specifies to apply the rule to those nets that only drive logic in the specified *to* domains and that are driven by logic from another power domain.

If multiple options are specified (excluding the `-force` option), the requirements specified with all options must be met.

- If you combine `-to` and `-from` options, the rule should apply to those nets that only drive logic in the specified *to* domains and that are driven by logic in the specified *from* domains.

- If you combine `-from` and `-pins` options, the rule should apply to those nets that drive or connect to the specified pins, and also meet the requirements of the `-from` option.

- If you combine `-to` and `-pins` options, the rule should apply to those nets that are driven by or connected to the specified pins, and also meet the requirements of the `-to` option.

- If you combine `-from`, `-to` and `-pins` options, the rule should apply to those nets that
  - Are driven by or connected to the specified pins
  - Only drive logic in the specified *to* domains
  - Are driven by logic in the specified *from* domains

How to Handle Rule for Nets Connected to Bidirectional Pins

A bidirectional pin must be treated as both an input pin and an output pin of the associated instance. The semantics of level shifter rules on nets connected to such special pins will then be handled just like other regular cases.

However, to avoid that the user makes mistakes due to the ambiguity of pin or port direction in the functional model, static verification tools (for example those checking the quality of a CPF specification) should issue an *error* if a bidirectional pin is referenced in both `-from` *and* `-to` options

Furthermore, the tool will issue a *warning* if bidirectional pins/ports are referenced in a level shifter rule to make the designer aware that the bidirectional ports may be from a blackbox and their direction will be resolved (and changed) after loading the correct module or cell definition.

When Can Implementation Tools Ignore A Level Shifter Rule

Unless the `create_level_shifter_rule` command was specified with the `-force` option, level shifter rules can be ignored in the following cases:

- The rule is specified for a floating net.
- The rule is specified for an undriven net.
- The specified net has its leaf level driver and loads belong to the same power domain.
- The rule is specified for a net connecting a top-level port and a pad-pin of a pad instance.

Level Shifter Insertion

You can specify the location for the level shifters in the update_level_shifter_rules command using the `-location` and the `-within_hierarchy` options.

The `-location` option specifies the power domain where to insert the level shifters.

- `from` inserts the level shifters in a hierarchy belonging to the originating power domain. Unless further constrained, the location will be the outermost logic hierarchy of the originating power domain.
- `to` inserts the level shifters inside a hierarchy belonging to the destination power domain. Unless further constrained, the location will be the outermost logic hierarchy of the destination power domain.
- `parent` inserts the level shifters in the logic hierarchy as follows:
  - If the rule is specified without the `-to` option, choose the parent hierarchy of the outermost logic hierarchy of the originating power domain.
  - If the rule is specified with the `-to` option, choose the parent hierarchy of the outermost logic hierarchy of the destination power domain

  Note: This is the only value allowed if the original rule was specified with the `-force` option.

- `any` (can only be used with the `-within_hierarchy` option) indicates that the level shifter can be inserted in any power domain to which the specified logic hierarchy belongs.

The location for the inserted level shifters can be further refined using the `-within_hierarchy` option. Unless otherwise specified, the logic hierarchy determined by `-location parent` or the logic hierarchy specified in `-within_hierarchy` must be voltage compatible with either the originating power domain or the destination power domain. Two domains are considered as voltage compatible if the two domains have the same nominal conditions or the same voltages in all modes.

Only level shifter cells with a `-valid_location` specification that is compatible with the insertion location can be used. If `-valid_location` is

- `from`—the power domain of the insertion hierarchy must be voltage compatible with the originating power domain
- `to`—the power domain of the insertion hierarchy must be voltage compatible with the destination power domain
- `either`—the power domain of the insertion hierarchy must be voltage compatible with either the originating or the destination power domain
- `any`—the power domain of the insertion hierarchy may belong to any power domain. This is the special case where the logic hierarchy specified through `-location parent` or `-within_hierarchy` does not have to be voltage compatible with either the originating power domain or the destination power domain.

If the `update_level_shifter_rules` command is specified with the `-cells` option, the verification tools will check that the specified cells are available in the library sets associated with the domain of the specified instance.

Isolation Rules

An isolation rule specifies one or more domain crossing which require isolation logic. Isolation logic is required when the leaf drivers and leaf loads of a net are in power domains that are not on and off at the same time, or because it is part of the design intent.

The targeted nets are selected by the options `-from`, `-to`, and `-pins` of the create_isolation_rule command.

This section covers the following topics:

Interpretation of Different Options

- `-force` specifies to apply the rule to the specified pin without any of the filtering based on driver and load analysis.

  Note: The `-force` option must be specified with the `-pins` option. The `-from` or `-to` option will be ignored if they are combined with the `-force` option.

- `-from` specifies to apply the rule to those nets driven by logic in the specified *from* domains and are driving logic in other power domains.

- `-to` specifies to apply the rule to those nets that only drive logic in the specified *to* domains and that are driven by logic from another power domain.

If multiple options are specified (excluding the `-force` option), the requirements specified with all options must be met.

- If you combine `-to` and `-from` options, the rule should apply to those nets that only drive logic in the specified *to* domains and that are driven by logic in the specified *from* domains.

- If you combine `-from` and `-pins` options, the rule should apply to those nets that drive or connect to the specified pins, and also meet the requirements of the `-from` option.

- If you combine `-to` and `-pins` options, the rule should apply to those nets that are driven by or connected to the specified pins, and also meet the requirements of the `-to` option.
- If you combine `-from`, `-to` and `-pins` options, the rule should apply to those nets that
  - Are driven by or connected to the specified pins
  - Only drive logic in the specified *to* domains
  - Are driven by logic in the specified *from* domains

How to Handle Rule for Nets Connected to Bidirectional Pins

A bidirectional pin must be treated as both an input pin and an output pin of the associated instance. The semantics of isolation rules on nets connected to such special pins will then be handled just like other regular cases.

However, to avoid that the user makes mistakes due to the ambiguity of pin or port direction in the functional model, static verification tools (for example those checking the quality of a CPF specification) should issue an *error* if a bidirectional pin is referenced in both `-from` *and* `-to` options.

Furthermore, the tool will issue a *warning* if bidirectional pins/ports are referenced in an isolation rule to make the designer aware that the bidirectional ports may be from a blackbox and their direction will be resolved (and changed) after loading the correct module or cell definition.

How to Handle Isolation Rules Created Without Isolation Condition

IP blocks can have special requirements for input ports. For example, when the signals driving these input ports are switched off, the signals must be held at specific values. For these signals, no isolation condition can be specified because the IP developer has no knowledge of how the IP will be used.

For such cases, there is a need to create isolation rules without enable conditions (neither `-isolation_condition` nor `-no_condition` option specified). Such isolation rules are called incomplete isolation rules.

On the other hand, a default isolation condition can be specified for a switchable power domain:

```
create_power_domain [-default_isolation_condition expression]
```

Note: An error will be given if a `-default_isolation_condition` option is specified for an unswitched domain—a domain created without `-shutoff_condition` or `-external_controlled_shutoff` option.

In case of an incomplete isolation rule the following restrictions apply:

- The target location for the isolation cell must be the from domain if the IP block is a macro cell.
- The rule can only be specified for the net that connects to an input port of a module or a macro cell.

To complete an incomplete isolation rule, tools need to check if the power domain containing the leaf level driver of the net selected by the rule has a default isolation condition. If a default isolation condition is defined, this condition can be used by the incomplete isolation rule to make the rule complete. If no default isolation condition was defined for the driving power domain, the incomplete isolation rule is treated as a design constraint.

How to Handle Isolation Rules Created With Clamp Value for Isolation Output

During RTL simulation, special consideration is required for isolation rule specified with `clamp_high` or `clamp_low`

1. The net must be considered corrupted if the signal value is `1` (`0`), and the isolation output is set to `clamp_low` (`clamp_high`) when the isolation enable becomes active during power down sequence and before the driving domain is shut off.
2. The clamp value will appear on the net when the driven domain is switched off.
3. The net must be considered corrupted if the signal value is1(0), and the isolation output is set to `clamp_low` (`clamp_high`) during power up sequence between the time when the power of the driving domain is restored and the time the isolation enable becomes inactive.

When inserting the isolation logic, the implementation tool must

1. Simply connect the output pin of the cell to the selected net without breaking the net into two net segments (unlike with regular isolation cells with both data input and output pins).
2. Consider only clamp type isolation cells when the `-cells` option is specified

When Can Implementation Tools Ignore An Isolation Rule

Unless the `create_isolation_rule` command was specified with the `-force` option, isolation rules can be ignored in the following cases:

- Isolation logic exists on the selected net and all the following conditions are met:
  - The isolation cell has the same cell type as requested by the rule
  - The isolation enable of the cell matches the isolation condition of the rule
- The rule is specified for a floating net.
- The rule is specified for an undriven net.
- The specified net has its leaf level driver and loads in the same power domain.
- The isolation condition is specified with virtual ports at the block level but the virtual ports do not have any port mapping at the top level.
- The rule is specified for a net connecting a top-level port and a pad-pin of a pad instance.

  If a rule is specified for a net connecting a top-level port and a pad-pin of a pad instance, the rule should not be implemented, regardless of the `-force` option.

Note: If the isolation rule (`create_isolation_rule`) is specified with the `-no_condition` option and the rule (`update_isolation_rules`) is specified with the `-cells` option and none of the specified cells are specified with the `-no_enable` option, the implementation tools should issue an error and not insert any isolation logic.

Note: If the isolation rule is specified with an isolation condition and the rule (`update_isolation_rules`) is specified with the `-cells` option and none of the specified cells are specified with the `-enable` option, the implementation tools should issue an error and not insert any isolation logic.

Isolation insertion

You can specify the location for the isolation logic in the update_isolation_rules command using the `-location` and the `-within_hierarchy` options.

The `-location` option specifies the power domain where to insert the isolation logic.

- `from` inserts the isolation logic in a hierarchy belonging to the originating power domain. Unless further constrained, the location will be the outermost logic hierarchy of the originating power domain.

- `to` inserts the isolation logic inside a hierarchy belonging to the destination power domain. Unless further constrained, the location will be the outermost logic hierarchy of the destination power domain.
- `parent` inserts the isolation logic in the logic hierarchy as follows:
  - If the rule is specified without the `-to` option, choose the parent hierarchy of the outermost logic hierarchy of the originating power domain.
  - If the rule is specified with the `-to` option, choose the parent hierarchy of the outermost logic hierarchy of the destination power domain

  Note: This is the only value allowed if the original rule was specified with the `-force` option.
- `any` (can only be used with the `-within_hierarchy` option) indicates that the isolation logic can be inserted in any power domain to which the specified logic hierarchy belongs.

The location for the inserted isolation logic can be further refined using the `-within_hierarchy` option. Only isolation cells compatible with the insertion location may be used. Compatibility is determined by the `-valid_location` option of the `define_isolation_cell` command. If `-valid_location` is

- `from`—this type of cell may only be used for `off` to `on` isolation. Since these cells rely on their primary power and ground pins for their normal function, they must be inserted into a power domain that is `on` whenever the source domain is `on`.
- `to`—this type of cell is used for `off` to `on` isolation and relies on its primary power and ground pins for its normal and isolation functions. Therefore, the domain of the insertion hierarchy must be `on` when the destination domain is `on`.
- `on`—this type of cell uses its primary power and ground pins for both the normal and isolation functions. For an
  - `on` to `off` isolation, the domain of the insertion hierarchy must be `on` when the originating domain is `on`.
  - `off` to `on` isolation, the domain of the insertion hierarchy must be `on` whenever the destination domain is `on`.
- `any`—these cells do not rely on their primary power or ground pins for their normal and isolation functions. Thus, there is no restriction on the power domain of the specified hierarchy.

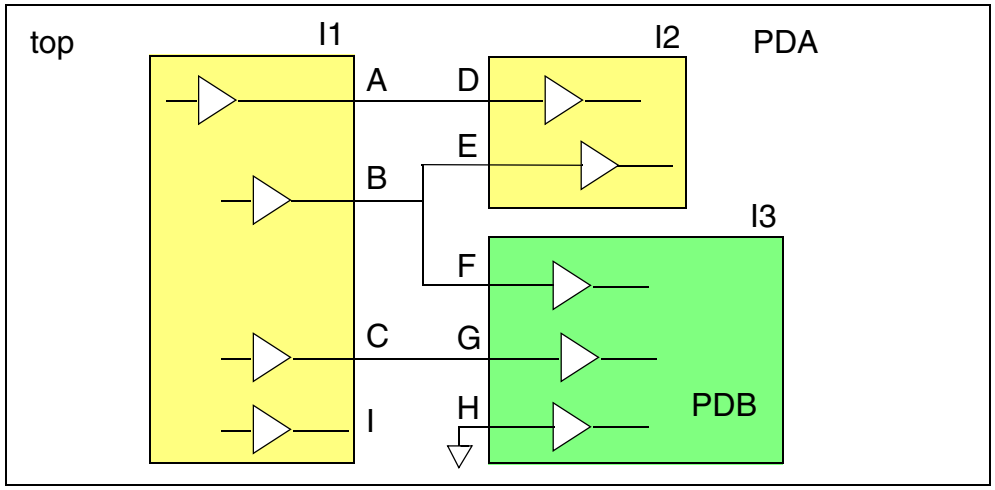
The other two values of `-valid_location`, `to` and `off`, are not commonly used and will be deprecated in a future release. All isolation cells can be described with the above three different valid locations.

if the `update_isolation_rules` command is specified with the `-cells` option, the verification tools will check that the specified cells are available in the library sets associated with the domain of the specified instance.

Example

The following example illustrates the effect of the specified options on the selection of the nets. In the design in Figure 6-1, instances `I1` and `I2` belongs to power domain `PDA`, and instance `I3` belongs to power domain `PDB`.

Figure 6-1 Effect of the Specified Options on the Nets Selected



The following table shows how the nets are selected based on the specified options:

Rules	Net Segments Selected
`create_isolation_rule -name iso1 -from PDA`	BF, CG
`create_isolation_rule -name iso2 -to PDB`	BF, CG
`create_isolation_rule -name iso3 -from PDA \ -to PDB -pins B`	BF

The isolation rules can be ignored in the following cases:

- Rule `iso1` can be ignored for the net connected to pin I in Figure 6-1 because it is a floating net.

- Rule `iso2` can be ignored for the net connected to pin `H` in Figure 6-1 because it is driven by a tie-low signal.
- Net segments `AD` and `BE` in Figure 6-1 should not be considered for any rule because the leaf-level driver and loads are in the same power domain.

Multiple Level Shifter and Isolation Rules

- How Do Multiple Rules Apply to a Physical Net
- How Do Rules Apply to Nets with Multiple Fanouts

How Do Multiple Rules Apply to a Physical Net

Multiple rules can be specified for a physical net.

In this case, the following policy should be used.

➤ The implementation tools should trace back from the net connected to the leaf level load pin to the net connected to the leaf level driver pin until it finds a net with a rule applied to it. That rule will be used.

Note: The verification tools should issue a warning if multiple rules are specified for the logical net segments of a physical net.

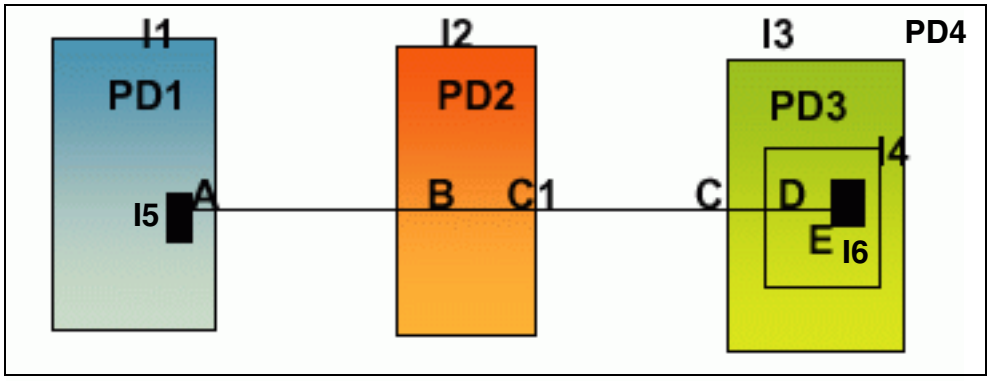
Note: If multiple isolation rules are specified for the logical net segments of a physical net and all rules use the same value for the `-isolation_target` option, then all isolation enable signals must be identical or must have been declared as equivalent control pins using the set_equivalent_control_pins command. Otherwise the verification tools should issue an error.

If a physical net has a valid level-shifter rule and a valid isolation rule, and the expected location of the cell for both rules is the same, the implementation tool will first try to find a matching cell (that can be used as both isolation cell and level shifter) to implement both rules. If no such cell can be found, the tool will need to implement the level shifter logic and isolation logic separately according to the rules.

Examples of Multiple Rules on a Physical Net

In the design in Figure 6-2 on page 103, instances `I1`, `I2`, `I3` belong to power domains `PD1`, `PD2`, `PD3`, respectively. The design has a physical net whose leaf driver (driving instance) is part of power domain `PD1` while the leaf load (driven instance) is part of power domain `PD3`. The logical net segment in `PD2` is a wire feedthrough.

Figure 6-2 Multiple Rules on a Physical Net



Example 6-1

Assume the following level shifter rules from the top-level design:

```
create_level_shifter_rule -name lsPD1 -from PD1
create_level_shifter_rule -name lsPD2 -pins I2/C* -from PD1
create_level_shifter_rule  -name lsPD3 -to PD3
```

In this case, rule `lsPD1` applies to the net connected to pin `I1/A`. Rule `lsPD2` applies to the net connected to pin `I2/C1`. Rule `lsPD3` applies to the net connected to pin `I3/I4/I6/E`.

All these pins are on the same physical net between pinsI1/A and I3/I4/I6/E. Because pin E is the leaf load pin of the physical net and the logical net connected to this pin has rule `lsPD3`, this rule will be used for level shifter insertion on this physical net and all other rules will be ignored for this physical net. See How Do Multiple Rules Apply to a Physical Net.

Example 6-2

Assume the following level shifter rules from the top-level design:

```
create_level_shifter_rule  -name lsPD1  -from PD1  -to PD3
create_level_shifter_rule  -name lsPD3 -to PD3
```

In this case, both rule `lsPD1` and rule `lsPD3` apply to the net connected to pin `I3/I4/I6/E`. However, rule `lsPD1` has a higher priority because it has both the `-from` and `-to` options, while rule `lsPD3` only has the `-to` option. Therefore, rule `lsPD1` takes precedence. For more information, see Different Categories of Rules.

Example 6-3

Assume the following CPF commands from the top-level design:

```
create_power_domain -name PD1 -default_isolation_condition iso
create_isolation_rule -name isoPD1 -to PD4 -isolation_condition iso1 \
  -isolation_output low
set_instance I3 -domain_mapping { {PD3 PD4} }
set_design mod3
...
create_isolation_rule -name isoPD3 -isolation_output high  -to PD3
end_design
```

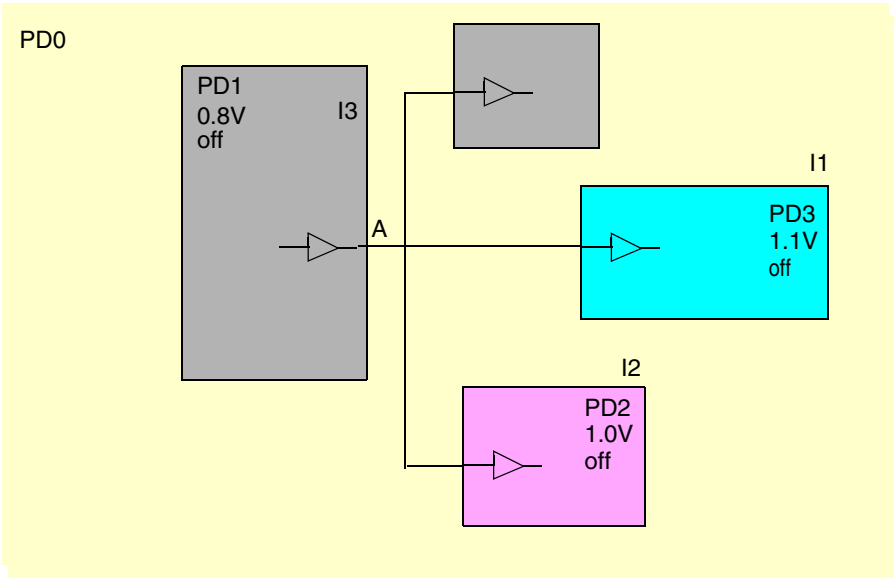
Block-level isolation rule `isoPD3` applies to `I4/I6/E` and is incomplete. According to How to Handle Isolation Rules Created Without Isolation Condition the tool will use the default isolation condition specified for the power domain of the leaf driver of `I3/I4/I6/E`. In this case, the leaf level driver is an instance in domain `PD1` which has a default isolation condition `iso`. In addition, rule `isoPD3` takes precedence over rule `isoPD1` based on the policy in Precedence of Rules in the Hierarchical Flow.

How Do Rules Apply to Nets with Multiple Fanouts

Special handling is required for level shifter and isolation cell insertion when the nets have multiple leaf-level loads and different rules apply for the nets connected to the loads.

The design in Figure 6-3 has three power domains. The default domain `PD1` operates at 0.8v. Power domains `PD2` and `PD3` operate at 1.0 and 1.1 volt, respectively.

Figure 6-3 Nets with Fanouts in Different Power Domains



The net connected to pin A fans out to different power domains that each operate at a different supply voltage. If a level shifter rule is applied to the net connected to pin `A` and the level shifter cell location is in the from domain (see Example 6-4), you need a different level shifter for each load and therefore pin A has to be cloned (see Figure 6-4 on page 106)

Example 6-4

Assume the following CPF commands from the top-level design:

```
create_power_domain -name PD0
create_power_domain -name PD1 --instances I3
create_power_domain -name PD2 -instances I2
create_power_domain -name PD3 -instances I1
create_level_shifter_rule -name lvl1 -from PD1
update_level_shifter_rule -names lvl1 -location from
```

To ensure that the verification tools can correctly verify the netlist with the cloned pins against the original HDL and the golden CPF, the implementation tools need to follow these rules:

1. Keep the original port with its original name.
2. Clone the ports.

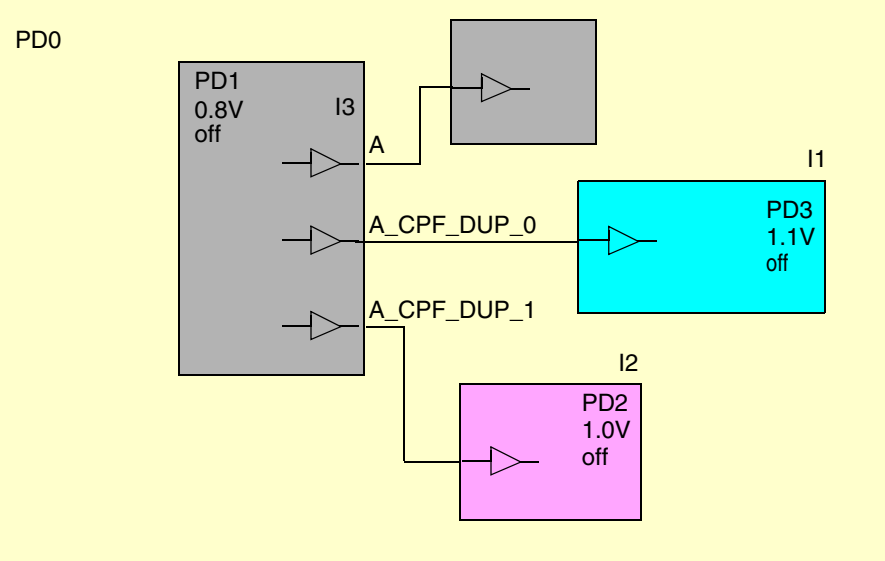
   Use the following format for the names of the cloned ports:

   *originalPortName*_CPF_DUP_*index*

   The index is an integer that starts at 0 and that is incremented as needed.
3. Apply the original isolation or level shifter rule to all cloned ports.

   Follow the precedence rules in Precedence of Rules in the Flat Flow, Precedence of Rules in the Hierarchical Flow, and How Do Multiple Rules Apply to a Physical Net to resolve any conflicts of rules specified on the physical nets connected to the original port and the cloned ports.

Figure 6-4 Cloned Ports for Nets with Fanouts in Different Power Domains



State Retention Rules

A state retention rule specifies one or more registers (at RTL) or instances in a switchable power domain that need to be mapped to state retention cells.

The targeted registers or instances are selected by the `-domain` and `-instances` option of the create_state_retention_rule command.

Interpretation of Different Options

- `-domain` specifies to apply the rule to all registers or non-state-retention sequential instances in this domain.
- `-instances` specifies a list of registers or instances

  See the description for the `-instances` option of the create_state_retention_rule command.

Note: The user needs to be aware that a state retention rule created with only the `-domain` option might result in unexpected behaviors during implementation. For example, new sequential instances (such as latches or flops) can be added during DFT synthesis. When the rule is applied to a synthesized netlist, the tools will replace such special sequential instances with retention cells. To avoid this from happening, designers should create a state retention rule with either the `-instances` or the `-exclude` option.

How to Model Different Types of State Retention Control

Note: In what follows, power refers to the primary power domain of the state retention logic. The secondary power domain of the state retention logic must be on for the save and restore operation to work properly. When the secondary power domain is off, the state value of the corresponding logic will be corrupted.

To model retention logic with only one retention control, you can use only the `-save_edge` and`-restore_edge` options.

- If you specify `-save_edge`, the register saves the current value when the expression changes to `true` and the power is on. The register restores the saved value when the power is turned on.
- If you specify `-restore_edge`, the register saves the current value when the expression changes from `true` to `false` and the power is on. The register restores the saved value when the expression changes from `false` to `true` and power is on.

- If you specify both options, the register saves the current value when the save expression changes from `false` to `true` and the power is on. The register restores the saved value when the restore expression changes from `false` to `true` and the power is on.

To model a retention logic with both save and restore control, specify both the `-save_level` and `-restore_level` options. The register saves the current value when the save expression is `true` and the power is on. The register restores the saved value when the restore expression is `true` and the power is on.

How to Handle State Retention Rules Created Without Any Control Condition

A state retention rule can be specified without any restore or save conditions. In this case, the rule is incomplete.

To complete an incomplete state retention rule, the tools need to check if the power domain to which the rule applies has a default save or restore condition.

- If a default save or restore condition is defined, this condition can be used by the incomplete state retention rule to make the rule complete.
- If no default save or restore condition was defined for the power domain, the incomplete state retention rule remains incomplete and will be ignored by the implementation tools.

Handling Master-Slave Type Retention Cells

A master-slave type state retention cell does not have a save or restore control pin. It has a secondary power or ground pin to provide continuous power supply to the slave latch. The clock signal acts as the retention control signal. As a result, users should use the clock signal for the registers as the restore edge and save edge.

If the clock signal is used as the restore or save signal for a flop, then the synthesis tool will synthesize it using a master-slave type retention cell. For library definition, the clock pin needs to be declared as the only save and restore signal and has to be an always-on pin.

Note: This type of cell has some challenges for implementation. For example, in retention mode, the clock needs to be at a specific value to keep the data in the slave latch. All logic on the clock tree must be always-on.

When Can Implementation Tools Ignore A State Retention Rule

State retention rules can be ignored in the following cases:

- The rule is incomplete

- The rule is specified for an unswitched domain or sequential elements in an unswitched domain.
  - If the `-required` option is specified, the retention logic cannot be optimized.
  - Implementation tools will typically not optimize the retention logic due to hierarchical flow considerations.
- There are no matching sequential elements in the domain.
- The specified instance is already an instantiation of a state retention cell.

Note: It is an error, if the virtual port used to specify a block-level retention control condition is not mapped to a top-level driver.

7

CPF File Structure

Command Categories

The following table shows how the CPF commands can be categorized.

Category	CPF Command
version command	set_cpf_version
hierarchical support commands	set_design
	set_instance
	end_design
	update_design
	get_parameter
macro support	set_instance
	set_macro_model
	end_macro_model
	set_analog_ports
	set_diode_ports
	set_floating_ports
	set_pad_ports
	set_power_source_reference_pin
	set_wire_feedthrough_ports
general purpose commands	find_design_objects
	include
	set_array_naming_style
	set_hierarchy_separator
	set_power_unit
	set_register_naming_style
	set_time_unit
verification support commands	assert_illegal_domain_configurations
	create_assertion_control
simulation support commands	set_sim_control

Category	CPF Command
mode and power mode commands	set_power_mode_control_group
	end_power_mode_control_group
	create_mode
	create_power_mode
	create_mode_transition
	update_power_mode
design and implementation constraints	create_analysis_view
	create_bias_net
	create_global_connection
	create_ground_nets
	create_isolation_rule
	create_level_shifter_rule
	create_nominal_condition
	create_operating_corner
	create_pad_rule
	create_power_domain
	create_power_nets
	create_power_switch_rule
	create_state_retention_rule
	define_library_set
	identify_always_on_driver
	identify_power_logic
	identify_secondary_domain
	set_equivalent_control_pins
	set_input_voltage_tolerance
	set_power_target
	set_switching_activity
	update_isolation_rules
	update_level_shifter_rules
	update_nominal_condition
	update_power_domain
	update_power_switch_rule

Category	CPF Command
	update_state_retention_rules
library-related commands	define_always_on_cell
	define_global_cell
	define_isolation_cell
	define_level_shifter_cell
	define_open_source_input_pin
	define_pad_cell
	define_power_clamp_cell
	define_power_clamp_pins
	define_power_switch_cell
	define_related_power_pins
	define_state_retention_cell

Tip

It is recommended to define the library-cell related commands in a separate file which can be sourced in the CPF file (using the include command) in a higher-level CPF file.

Typical Command Usage

To perform functional verification, you need the following minimum set of commands to specify the power structures when starting from RTL:

```
set_design
end_design
set_macro_model
end_macro_model
create_power_domain
create_nominal_condition
create_power_mode
create_state_retention_rule
create_isolation_rule
```

To drive synthesis and test, you need at least the following commands in addition to the commands listed above:

```
define_library_set
define_isolation_cell
define_level_shifter_cell
define_state_retention_cell
update_nominal_condition
update_power_mode
```

To drive physical implementation and signoff analysis, you need at least the following commands in addition to all commands listed above:

```
create_power_nets
create_ground_nets
create_power_switch_rule
update_power_domain
create_operating_corner
create_analysis_view
```

Command Dependency

Some CPF commands reference objects defined in previous commands. This implies that a certain command order is required. If this order is violated, an error will be issued.

CPF Command	Defines	References	Order
set_cpf_version	version	--	first (if specified)
include	other CPF specification		can be anywhere
set_array_naming_style set_hierarchy_separator set_register_naming_style	name mapping of design objects		after set_design and before any commands that reference design objects
set_power_unit	power unit		after set_design
set_time_unit	time unit		after set_design
define_library_set	library set	library	after set_cpf_version (if specified)
define_always_on_cell	always on buffer		if needed, after define_library_set
define_global_cell	global cell		if needed, after define_library_set
define_isolation_cell	isolation cell		if needed, after define_library_set, before update_isolation_rules
define_level_shifter_cell	level shifter		if needed, after define_library_set
define_open_source_input_pin	open source input pin		if needed, after define_library_set
define_power_clamp_cell	power clamp cell		if needed, after define_library_set
define_power_switch_cell	power switch		if needed, after define_library_set
define_state_retention_cell	retention cell		if needed, after define_library_set, before update_state_retention_rules
define_pad_cell	pad cell		if needed, after define_library_set
define_power_clamp_pins	cell pin with built-in clamp		if needed, after define_library_set
define_related_power_pins	relationship between power and data pins		if needed, after define_library_set
set_design		design or module	after library-related specifications, but before design-related specifications related to top design
set_instance		hierarchical instance	
find_design_objects			after set_design
get_parameter		parameters in set_design	after set_design

CPF Command	Defines	References	Order
create_power_domain	power domain		after set_design
create_nominal_condition	nominal condition		after set_design
set_power_mode_control_group	power mode control group	power domain	after create_power_domain
end_power_mode_control_group	end of power mode control group definition		
create_mode	generic mode	power domain nominal condition	after create_power_domain and create_nominal_condition
create_power_mode	power mode	power domain nominal condition	after create_power_domain and create_nominal_condition
assert_illegal_domain_configurations	illegal power mode	power domain nominal condition	after create_power_domain and create_nominal_condition
create_isolation_rule	isolation rule	power domain	after create_power_domain
create_level_shifter_rule	level shifter rule	power domain	after create_power_domain
create_pad_rule	pad rule	power domain	after create_power_domain
create_ground_nets	ground net	ground net	after set_design
create_power_nets	power net	power net	after set_design
create_power_switch_rule	power switch rule	power domain power net ground net	after create_power_domain and create_power_nets and create_ground_nets
create_state_retention_rule	state retention rule	power domain	after create_power_domain
create_mode_transition	transition mode	power mode	after create_power_mode
create_assertion_control			after create_power_domain
set_sim_control			after create_power_domain and create_xx_rule
identify_always_on_driver	always on driver		
identify_power_logic	isolation logic		after set_design
identify_secondary_domain	secondary domain		after set_design
set_equivalent_control_pins	equivalent pins	pins	after create_power_domain, create_isolation_rule and create_state_retention_rule
set_switching_activity	activities	pins	after set_design
create_operating_corner	corner		after define_library_set
create_analysis_view	analysis view	view	after create_power_mode and create_operating_corner
set_power_target			after set_power_unit
set_macro_model	custom IP		

CPF Command	Defines	References	Order
set_floating_ports		ports	after set_macro_model
set_input_voltage_tolerance	bias voltage	ports	after set_macro_model
set_wire_feedthrough_ports	physical connection	ports	after set_macro_model
set_analog_ports	analog port	ports	after set_macro_model
set_diode_ports	diode port	ports	after set_macro_model
set_pad_ports	pad port	ports	after set_macro_model
set_power_source_reference_pin	voltage reference pin	ports	after set_macro_model
end_macro_model	end of macro definition		
update_design			
update_nominal_condition		nominal condition	after create_nominal_condition
update_power_domain		power domain	after create_power_domain
update_power_mode			after create_power_mode
update_isolation_rules			after create_isolation_rule
update_level_shifter_rules			after create_level_shifter_rule
update_power_switch_rule			after create_power_switch_rule
update_state_retention_rules			after create_state_retention_rule
create_bias_net	bias net	bias net	after set_design
create_global_connection		power domain bias net power net ground net	depending on nets connected and options used, after create_power_domain, create_bias_net, create_ground_nets, create_power_nets
end_design	end of CPF file		last command in each CPF file.

Information Precedence

- If you define a CPF object in a specific scope multiple times with the same name, the last definition takes precedence, unless specified otherwise.

  For example, assume you see the following constraints within the same scope:

  ```
  create_power_domain -name PD1 -instances {A B}
  create_power_domain -name PD1 -instances {A }
  ```

  The last definition prevails and only instance A is added to power domain PD1.

  In the following example, the nominal condition `change` is defined first, next used in the power mode definition of mode `M1`, then redefined and used again in the power mode definition of `M2`.

  ```
  create_nominal_condition -name high -voltage 1.2
  create_nominal_condition -name low -voltage 1.0
  create_nominal_condition -name change -voltage 0
  create_power_mode -name M1 -domain_conditions {PD1@high PD2@change PD3@low}
  create_nominal_condition -name change -voltage 1.2
  create_power_mode -name M2 -domain_conditions {PD1@high PD2@change PD3@off}
  ```

  Because the last definition in the scope takes precedence, the last definition of nominal condition `change` will be used and applied to both power mode definitions (`M1` and `M2`).

- You can add implementation details for CPF objects using multiple update commands as long as each command specifies unique information. If the same information is specified, the information specified in the last command takes precedence.

  For example, consider the following 3 commands.

  ```
  update_power_domain -name PD1 -primary_power_net VDD1
  update_power_domain -name PD1 -pmos_bias_net VDD_bias
  update_power_domain -name PD1 -primary_power_net VDD2
  ```

  The result of these commands is that VDD2 is considered as the primary power net for power domain PD1. The second command specifies that power domain PD1 has a body bias net for the p-type transistors of all functional gates in power domain PD1.These 3 commands combined give the same result as the following command:

  ```
  update_power_domain -name PD1 -primary_power_net VDD2 -pmos_bias_net VDD_bias
  ```

- Similarly, you can add specification details for special library cells using multiple define commands, as long as each command specifies unique information. If the same information is specified, the information specified in the last command takes precedence.

  For example, the isolation cell definition for cell `ISO1` will include all the information from both `define` commmands.

  ```
  define_isolation_cell -cells {ISO1 ISO2 ISO3}  -enable en -location to
  define_isolation_cell -cells {ISO1} -always_on_pins {en}
  ```

- If information defined in the CPF file conflicts with information in the referenced library, the information in the CPF file takes precedence.

Information Inheritance

The general purpose commands are scope sensitive:

```
set_array_naming_style
set_cpf_version
set_hierarchy_separator
set_register_naming_style
set_time_unit
set_power_unit
```

By default, the scope inherits the values of the previous scope.

You can change the values for the current scope, but these values only apply as long as you are within the scope.

Example

Assume the following design hierarchy:

```
Top
    A_inst
        A1_inst
```

```
# the following command sets the hierarchy separator to / for all scopes
set_hierarchy_separator /
set_design Top
# the following command changes the hierarchy separator from / to . for Top
set_hierarchy_separator .
set_instance A_inst
set_design A
...
# the current scope inherits . as the hierarchy separator
# the following command changes the hierarchy separator to / for the current scope
set_hierarchy_separator /
...
end_design
# the scope changes to Top and the hierarchy separator changes back to .
# because . is the hierarchy separator for Top
...
set_instance A_inst.A1_inst
# the following command changes the hierarchy separator to / for the current scope
set_hierarchy_separator /
set_design A1
...
end_design
# the scope changes to Top and the hierarchy separator changes back to .
# because . is the hierarchy separator for Top
...
end_design
```

Object References

CPF commands reference design and CPF objects. An object reference is always relative to the current scope (part of the design that is visible).

By default, the scope starts at the top design and by default all elements in the design hierarchy are visible. You can change the scope using the set_instance command. After a `set_instance` command, the scope is reduced to a portion of the design.

You can only reference objects within the current scope or in a lower scope. The current scope is always considered to be the top. Top level does not necessarily refer to the top level of the design. In the current scope, top refers to the topmost level of the hierarchy of the scope.

To reference an object that is part of the current scope, you can use its name.

To reference an object that is below the current scope, specify the hierarchical name (path to the object). The object reference starts from the top of the scope. Therefore you can omit the hierarchical separator at the beginning of a path which denotes the top of the current scope.

Note: If your current scope is the root-level hierarchy and the root-level hierarchy has multiple top modules, `t`he search for design objects starts from all top modules.

Referencing Design Objects

Referencing a Pin or Port

Whenever you reference a pin or port in an expression, either

- The pin or port must exist in the design
- The port must have been declared as a virtual port using the `-ports,-input_ports`, `-output_ ports`, or `-inout_ports` option of the set_design command.

Referencing a Power Supply Net

When you reference a power supply net, either

- The net must exist in the design
- The net must have been created using the create_bias_net, create_ground_nets, or create_power_nets command.

Referencing RTL registers

To reference RTL registers, use the RTL register name. For arrays of registers, you can also use indexes to select bits of the array.

Examples

Assume the following RTL:

```
reg [5:4][3:2] c;
reg c1;
```

- The following command will select register `c1` and the entire array of `c`.

  ```
  create_state_retention_rule -name ret -instances c*
  ```

  As illustrated by this example, the wildcard can match bus delimiters.

- The following command will select the entire array of `c`.

  ```
  create_state_retention_rule -name ret -instances c
  ```

- The following command selects registers `c[5][3]` and `c[5][2]`:

  ```
  create_state_retention_rule -name ret -instances {c[5][*]}
  ```

- The following command selects registers `c[5][3]` and `c[4][3]`:

  ```
  create_state_retention_rule -name ret -instances {c[*][3]}
  ```

Referencing Verilog Generated Instances

To reference a Verilog generated instance, follow the Verilog-2001 naming style for generated instances.

Example

Assume the following RTL:

```
parameter SIZE = 2;
genvar i, j, k, m;
generate
    for (i=0; i<SIZE+1; i=i+1) begin:B1 // scope B1[i]
      M1 N1(); // instantiates B1[i].N1
      for (j=0; j<SIZE; j=j+1) begin:B2 // scope B1[i].B2[j]
      M2 N2(); // instantiates B1[i].B2[j].N2
        for (k=0; k<SIZE; k=k+1) begin:B3 // scope B1[i].B2[j].B3[k]
        M3 N3(); // instantiates B1[i].B2[j].B3[k].N3
        end
      end
      if (i>0)
        for (m=0; m<SIZE; m=m+1) begin:B4 // scope B1[i].B4[m]
        M4 N4(); // instantiates B1[i].B4[m].N4
      end
    end
endgenerate
```

The generated instance names that should be used are:

- `B1[0].N1, B1[1].N1`
- `B1[0].B2[0].N2, B1[0].B2[1].N2`
- `B1[0].B2[0].B3[0].N3, B1[0].B2[0].B3[1].N3, B1[0].B2[1].B3[0].N3`
- `B1[1].B4[0].N4, B1[1].B4[1].N4`

The following command references the registers in the generated instances `B1[0].N1` and `B1[1].N1`.

```
create_state_retention_rule -name sr1 ... -instances {B1[*].N1.q*}
```

Referencing an Instance

When you reference a hierarchical instance, the instance and all of its children will be selected.

Consider the following rule for the following hierarchy

```
create_state_retention_rule -name sr1 -instances A/A* -exclude A/A3
```

```
Top
    A
      A1
        flop
      A2
        flop
      A3
        flop
      B
```

The `-instances` option selects instances `A1`, `A2` and `A3` in hierarchical instance `A`. However since `A1`, `A2` and `A3` are hierarchical instances themselves, this command selects the leaf-level flops in `A1`, `A2` and `A3`. The `-exclude` option removes `A/A3/flop` from the selected list.

Referencing CPF Objects

- You can only reference a CPF object that was already created.
- To reference CPF object created *in* the current scope, you can use the same name.
- To reference a CPF object created *below* of the current scope, use the hierarchical name of the CPF object. This is the defined name of the CPF object prefixed with the hierarchical name of the scope in which the CPF object is created with respect to the current scope.
- All CPF objects except for the library set are scope sensitive.
- Library set objects are defined in a global non-hierarchical namespace. Name conflicts are resolved according to Information Precedence.

Assume a design with the following hierarchy:

```
Top
    Inst1 (mod1)
    Inst2 (mod2)
      Inst3 (mod3)
      Inst4 (mod4)
```

The names in parentheses are the corresponding module names. Consider the following CPF file:

```
1.  set_design Top
2.  create_power_domain -name PD1 -default
3.  create_power_domain -name PD2 -instances Inst1
4.  set_instance Inst2
5.  set_design mod2
6.  create_power_domain -name PD1 -instances Inst3
7.  create_isolation_rule -name ir1 -from PD1 -isolation_condition standby1
8.  create_isolation_rule -name ir2 -from PD2 -isolation_condition standby2
9.  create_power_domain -name PD3 -instances Inst4
10. end_design
11. create_isolation_rule -name ir3 -from PD1 -isolation_condition standby3
12. create_level_shifter_rule -name lsr1 -from Inst2.PD1
13. create_state_retention_rule -name srr1 -domain PD3
14. end_design
```

In this CPF file, two power domains with the same name `PD1` are created (lines 2 and 6). This is allowed because they are created in two different scopes.

When referencing power domain `PD1` inside scope `Inst2` (line 7), the only matching power domain is the power domain `PD1` created on line 6.

Referencing power domain `PD2` on line 8 causes an error, because there is no power domain created with name `PD2` in the scope of `Inst2`.

Referencing power domain `PD3` on line13 also causes an error because power domain `PD3` was not created at the top level, even though a power domain `PD3` was created inside scope `Inst2` (line 9).

To reference power domain `PD1` defined for scope `Inst2` from the top level, the hierarchical path name of object `PD1` with respect to the current scope (top level) must be used (line 12).

8

General CPF Commands

assert_illegal_domain_configurations

```
assert_illegal_domain_configurations -name mode
     { -domain_conditions domain_condition_list
     | -group_modes group_modes_list
     | -domain_conditions domain_condition_list -group_modes group_mode_list }
```

Asserts that a particular configuration of domain conditions and power mode control group conditions is illegal. The assertion is only checked against conditions explicitly specified. No assumptions are made about unspecified domains.

A verification tool will flag if the design enters a power mode that matches all of the domain and group conditions specified in this command.

An error message is issued if the design operates in a configuration of power domain conditions that have not been defined in a power mode.

Note: Verification tools must issue an error message for illegal power modes, but implementation tools might use these power modes to permit optimization.

Options and Arguments

`-domain_conditions` *`domain_condition_list`*

Specifies the nominal condition of each power domain to be considered an illegal configuration.

Use the following format to specify a domain condition:

`domain_name``@`*`nominal_condition_name`*

Note: You can associate each power domain with only one nominal condition per command.

The power domains must have been previously defined with the `create_power_domain` command.

The condition must have been previously defined with the `create_nominal_condition` command.

`-group_modes` *`group_mode_list`*

Specifies the mode of each power mode control group.

Use the following format to specify the mode for a group:

`group_name``@`*`power_mode_name`*

The specified *`group_name`* refers to the name of a power mode control group, previously defined with the `set_power_mode_control_group` command or to the name of the scope in case the power control group was automatically defined. The specified *`power_mode_name`* must have been defined in a `create_power_mode` command contained in the definition of the specified power mode control group.

`-name` *`string`*

Specifies the name of the mode.

This power mode *cannot* be defined with the `create_power_mode` command.

Note: The specified string cannot contain wildcards nor the hierarchy delimiter character.

Example

The following command declares that it is illegal for power domains `PD1` and `PD2` to have the nominal condition `a` at the same time.

```
assert_illegal_domain_configurations -name foo -domain_conditions {PD1@a PD2@a}
```

Related Information

Power Mode on page 18

create_nominal_condition on page 159

create_power_domain on page 168

create_power_mode on page 177

set_power_mode_control_group on page 237

Power Mode Control Groups on page 81

create_analysis_view

```
create_analysis_view
     -name string
     -mode mode
     { -domain_corners domain_corner_list
     | -group_views group_view_list
     | -domain_corners domain_corner_list -group_views group_view_list}
     [ -user_attributes string_list] [-default]
```

Creates an analysis view. Associates a list of operating corners with a given mode.

The set of active analysis views represents the different design variations (MMMC, that is, multi-mode multi-corner) that will be timed and optimized.

A power design can have no analysis views, but if any are specified, one and only one can be the default. If you do not specify the default, the first view defined will be designated as the default.

Options and Arguments

`-default`

Designates the specified view as the default view.

Note: For a given scope you can only have one analysis view as default.

`-domain_corners` *`domain_corner_list`*

Specifies the operating corner of the power domain to be considered in the specified mode.

Use the following format to specify a domain corner:

`domain_name@corner_name`

Specify a corner for each domain to be considered for the specified power mode.

Note: You must also include the corner definition for a domain that is switched off in this view.

The power domain must have been previously defined with the `create_power_domain` command.

The corner must have been previously defined with the `create_operating_corner` command.

`-group_views` *`group_view_list`*

Specifies the view of each power mode control group to be considered with the specified analysis view.

Use the following format to specify the mode for a group:

`group_name``@`*`view_name`*

The specified *`view_name`* must have been defined in a previous `create_analysis_view` command contained in the definition of the specified power mode control group (`set_power_mode_control_group`).

`-mode` *`mode`*

Specifies a mode.

The mode must have been previously defined with the `create_power_mode` command.

`-name` *`string`*

Specifies the name of the analysis view.

Note: The specified string cannot contain wildcards nor the hierarchy delimiter character.

`-user_attributes` *`string_list`*

Associates a list of user-defined attributes with the analysis view. Specify a list of strings.

Example

The following example applies to a design with three power domains (PD1, PD2, and PD3). The design can operate in two modes (M1, M2).

The following table shows the nominal voltages for each power domain in each of the modes.

Power Mode	**Power Domain**		
	PD1	**PD2**	**PD3**
M1	1.2V	1.2V	1.0V
M2	1.2V	1.0V	0.0V

The best and worst corner voltage of each mode is +10 percent and -10 percent of the nominal voltage for a domain in that mode.

As a result the operating corner and the analysis view can be described as below.

```
create_operating_corner -name high_max -voltage 1.3 -library_set lib_1.3
create_operating_corner -name low_max -voltage 1.1 -library_set lib_1.1
create_operating_corner -name high_min -voltage 1.1 -library_set lib_1.1
create_operating_corner -name low_min -voltage 1.0 -library_set lib_1.0
create_operating_corner -name off -voltage 0.0 -library_set lib_1.0
create_analysis_view -name fast_M1 -mode M1 \
-domain_corners {PD1@high_max PD2@high_min PD3@low_max}
create_analysis_view -name slow_M1 -mode M1 \
-domain_corners {PD1@high_min PD2@high_min PD3@low_min}
create_analysis_view -name fast_M2 -mode M2 \
-domain_corners {PD1@high_max PD2@low_max PD3@off}
```

Related Information

Analysis View on page 15

create_operating_corner on page 163

create_power_mode on page 177

create_assertion_control

```
create_assertion_control
     -name string
     { -assertions assertion_list
     | -domains power_domain_list}
     [ -exclude assertion_list]
     [ -shutoff_condition expression]
     [ -type {reset | suspend} ]
```

Inhibits evaluation of any selected assertion instance when its related power domain is powered down.

By default, assertions remain active when the power domain is powered down.

Note: If the selected assertion instance belongs to an unswitched domain, the evaluation is not affected.

Methodology Implications

1. An assertion instance is related to a power domain if it is associated with a hierarchical (module) instance that belongs to that power domain. An assertion instance is associated with a hierarchical (module) instance if it is bound to that instance, appears in that instance, or is bound to the module definition for that instance. Each assertion instance is associated with a unique power domain.
2. An assertion that monitors the behavior of signals that are driven by logic entirely within a single power domain should be contained within, or bound to an instance of, the module containing that logic.
3. An assertion that monitors the behavior of signals that are driven by logic contained in two or more power domains should be contained within, or bound to an instance of, a module associated with a power domain that is always on when any of the monitored power domains is on.
4. An assertion that appears in or is bound to a testbench will be considered to be in an unswitched power domain.

Important

Tools are not required to check that these methodology guidelines are followed. Designers need to make sure that assertions are created for the right power domain to check the expected behavior.

Options and Arguments

`-assertions` *`assertion_list`*

Selects only the assertions whose names are included in the assertion list.

Assertion names can contain wildcards.

`-domains` *`power_domain_list`*

Selects all assertions that appear in any hierarchical instance associated with one of the specified power domains.

The power domains must have been previously defined with the `create_power_domain` command.

`-exclude` *`assertion_list`*

Specifies to exclude the specified list of assertions from the list of already selected assertions through the `-assertions` or `-domains` option.

`-name` *`string`* Specifies the name of the assertion control.

Note: The specified string cannot contain wildcards nor the hierarchy delimiter character.

`-shutoff_condition` *`expression`*

Specifies the condition when the selected assertions should be disabled.

By default, the selected assertions are disabled when the associated power domain of the assertion is powered down.

`-type {reset|suspend}`

Specifies whether to reset or suspend the assertions when the shutoff condition evaluates from `true` to `false`. The shutoff condition of a power domain controls evaluation of any selected assertion in that power domain.

If `type` is `reset`, all state information is cleared and the assertions will be evaluated from the reset state.

If `type` is `suspend`, evaluation of controlled assertions continues from where it left off previously.

Default: `reset`

Examples

In the following examples, a `set_hierarchy_separator` command has been included in the CPF file to specify that the slash (/) is the hierarchy separator.

- In the following example, the `create_assertion_control` command suppresses the evaluation of two assertion instances associated with hierarchical instance `add_ef`. Instance `add_ef` belongs to power domain `PD_add_ef`. The assertions are turned off when the power domain shuts off. The `-type reset` option is included to specify that the assertions should be reset. Because `reset` is the default behavior, the `-type` option could be omitted.

```
create_power_domain -name PD_add_ef -instances {add_ef} \
    -shutoff_condition {u_pmc/pso_en[2] & u_pmc/cond_3[2]}

create_assertion_control -name ac1 \
    -assertions {add_ef/SVA_A1 add_ef/SVA_A2} \
    -type reset
```

- In the following example, the `create_assertion_control` command suppresses the evaluation of all assertion instances associated with all instances included in the power domains `PD_add_ab` and `PD_mux`.

```
create_power_domain -name PD_add_ab -instances {add_ab} \
    -shutoff_condition {u_pmc/pso_en[0] & u_pmc/cond_3[0]}
create_power_domain -name PD_mux -instances {mux} \
    -shutoff_condition {u_pmc/pso_en[3] & u_pmc/cond_3[3]}
...
create_assertion_control -name ac1 -domains {PD_add_ab PD_mux}
```

- The following example includes the `-type suspend` option. When power domain `PD_add_ef` is powered down, the specified assertions are turned off. When the power domain is powered up, the assertions are in the same state they were in at power down, and evaluation continues.

```
create_power_domain -name PD_add_ef -instances {add_ef} \
    -shutoff_condition {u_pmc/pso_en[2] & u_pmc/cond_3[2]}
...
create_assertion_control -name ac2 \
    -assertions {add_ef/SVA_A1 add_ef/SVA_A2} \
    -type suspend
```

- The following example specifies the condition when the selected assertions should be disabled.

```
create_assertion_control -name ac1 \
    -assertions {add_ab/SVA*} \
    -shutoff_condition {!u_pmc/pso_en[0] & !u_pmc/cond_3[0]}\
    -type reset
```

Related Information

create_power_domain on page 168

create_bias_net

```
create_bias_net
     -net net
     [-driver pin]
     [-user_attributes string_list]
     [-peak_ir_drop_limit float]
     [-average_ir_drop_limit float]
```

Specifies or creates a bias net to be used as a power supply to either forward or backward body bias a transistor.

Note: Even if this net exists in the RTL or the netlist, it still must be declared through this command if the net is referenced in other CPF commands.

You can use the create_global_connection command to connect the net to the appropriate pins during physical implementation.

Options and Arguments

`-average_ir_drop_limit` *float*

Specifies the maximum allowed average voltage change on a bias net due to resistive effects in volt (V) for any mode.

Default: 0

`-driver` *pin*

Specifies the driver of the net. Specify a port or instance pin.

`-net` *net*

Declares a bias net.

Note: You can specify a hierarchical net name to infer a bias net in another scope.

`-peak_ir_drop_limit` *float*

Specifies the maximum allowed peak voltage change on a bias net due to resistive effects in volt (V) for any mode.

Default: 0

`-user_attributes` *string_list*

Attaches a list of user-defined attributes to the net. Specify a list of strings.

Example

- The following command declares the net `bVdd` as a bias net driven by pin BVDD.

```
create_bias_net -net bVdd -driver BVDD
```

Related Information

create_global_connection on page 140

create_global_connection

```
create_global_connection
     -net net
     { -pins pin_list | -ports port_list | -pg_type pg_type_string }
     [-domain power_domain | -instances instance_list]
```

Specifies how to connect a global net to the specified pins. A global net can be a data net, bias net, power net or ground net.

Given a list of pins, if a specified pin is already connected in Verilog, that pin is ignored for connection. For pins that are not connected in Verilog, the following priority determines the pin connection with the specified net:

- A connection defined with `-instance` has a higher priority than a connection specified with `-domain`.
- If the same pin is specified with multiple connections and all specifications are with `-instance` (or all specifications are with `-domain`), the last definition wins.

This command allows to specify which pins must be connected. You can

- Specify all pins to be connected with the `-pins` option

  If you omit the `-domain` or `-instances` option, the global connection applies to the specified pins of the entire design.
- Combine options to filter the set of pins:
  - Combine `-pins` and `-domain` options—only connects those pins in the specified list that also belong to the specified power domain
  - Combine `-pins` and `-instances` options—only connects those pins in the specified list that also belong to the specified instances.

Options and Arguments

`-domain` *power_domain*

Limits the pins to be connected to pins that belong to the specified power domain.

The power domain must have been previously defined with the `create_power_domain` command.

`-instances` *instance_list*

Limits the pins to which the specified global net should be connected to pins that belong to the specified instances. Specify the name with respect to the current design or top design.

You can use wildcards (*) to specify a pattern of instance names.

`-net` *net*

Specifies the name of the global net for which you specify the global connection.

If the specified net does not exist in the design, you must have defined it with a `create_bias_net`, `create_power_nets` or `create_ground_nets` command.

`-pg_type` *pg_type_string*

Identifies the instance pins to connect to the specified global net by selecting those pins whose `pg_type` attribute in the corresponding Liberty cell pin definition matches the *pg_type_string*.

No wildcard characters are allowed.

`-pins` *pin_list*

Specifies the name of the LEF pin to connect to the specified global net.

If several pins of the same instance have names that match the specified names, all those pins will be connected to the specified global net.

You can use wildcards (*) to specify the pin names.

`-ports` *port_list*

Specifies the names of top-level module ports to connect to the specified global net.

You can use wildcards (*) to specify the port names.

Examples

- The following command defines the global net connection for net `vdd1`. All pins with name `VDD` in the design will be connected.

```
create_global_connection -net vdd1 -pins VDD
```

- The following command defines the global net connection for net `vdd2` in power domain PD2. All pins with name `VDD` in power domain `PD2` will be connected.

```
create_global_connection -net vdd2 -domain PD2 -pins VDD
```

- The following command defines the global net connection for net `vddc` to pin `VDDC of` instance `srpg1` inside hierarchical instance `A`.

  ```
  create_global_connection -net vddc -pins VDDC -instances A.srpg1
  ```

- The following command defines the global net connection for net `vddc` to pin `VDDC of` all leaf instances in hierarchical instance `a.b.c` starting with `i,` and of all leaf instances in hierarchical instances `a.b.c.*` starting with `i`.

  ```
  create_global_connection -net vddc -pin VDDC \
  -instances {a.b.c.i* a.b.c.*.i*}
  ```

  This command would connect net `vddc` to pin `VDDC` of all leaf instances such as

  ```
  a.b.c.i1
  a.b.c.i2
  a.b.c.d.i1
  a.b.c.e.i1
  ```

  but would not connect to

  ```
  a.b.c.d.e.i
  a.b.c.d.f.i
  ```

  because the wildcard does not include the hierarchical separator.

Related Information

Object Lists on page 25

create_bias_net on page 138

create_ground_nets on page 143

create_power_domain on page 168

create_power_nets on page 182

create_ground_nets

```
create_ground_nets
     -nets net_list
     [-voltage {float | voltage_range}]
     [-external_shutoff_condition expression | -internal]
     [-user_attributes string_list]
     [-peak_ir_drop_limit float]
     [-average_ir_drop_limit float]
```

Specifies or creates a list of ground nets.

Note: Even if this net exists in the RTL or the netlist, it still must be declared through this command if the net is referenced in other CPF commands.

The ground nets are created within the current scope.

Options and Arguments

`-average_ir_drop_limit` *float*

Specifies the maximum allowed average ground bounce on the specified ground nets due to resistive effects in volt (V) for any mode.

Default: `0`

`-external_shutoff_condition` *expression*

When a specified ground net is connected to an external source, you can use a Boolean expression to specify under which condition the external source can be switched off.

In this case, the net must be connected to an I/O port or pad.

Note: If neither this option nor the `-internal` option is specified, the ground source is assumed to be the primary source of an unswitched power domain.

`-internal`

Specifies that the nets must be driven by an on-chip ground switch. You must specify this option if this ground net is the primary ground net of an internal switchable power domain using ground (footer) switches.

`-nets` *`net_list`*

Declares a list of ground nets.

Note: You can specify a hierarchical net name to infer a ground net in another scope.

`-peak_ir_drop_limit` *`float`*

Specifies the maximum allowed peak ground bounce on the specified grounds net due to resistive effects in volt (V) for any mode.

Default: `0`

`-user_attributes` *`string_list`*

Attaches a list of user-defined attributes to the net. Specify a list of strings.

`-voltage {`*`float`* | *`voltage_range`*`}`

Identifies the voltage applied to the specified nets.

The voltage range must be specified as follows:

`lower_bound:upper_bound`

Specify the lower bound and upper bound, respectively.

Example

- The following command declares the net `vss` as a global ground net.

```
create_ground_nets -nets vss
```

create_isolation_rule

```
create_isolation_rule
     -name string
     [-isolation_condition expression | -no_condition]
     { -force -pins pin_list
     | -from power_domain_list | -to power_domain_list
     | -from power_domain_list -to power_domain_list }
     [-pins pin_list]
     [-exclude pin_list]
     [-isolation_target {from|to}]
     [-isolation_output { low | high | hold | tristate |clamp_high | clamp_low}
     [-isolation_control list_of_additional_controls]
     [-secondary_domain power_domain]
```

Defines a rule for adding isolation cells.

This command allows to indicate which domain crossings must be isolated. You can

- Select all domain crossings driven by logic in the domains specified with the `-from` option *and* that are driving logic in other power domains
- Select all domain crossings driving logic in the domains specified with the `-to` option *and* that are driven by logic from another power domain.
- Combine options to filter the set of domain crossings:
  - If you combine `-to` and `-from` options, the rule should apply to those domain crossings that only drive logic in the specified *to* domains and that are driven by logic in the specified *from* domains.
  - If you combine `-from` and `-pins` options, the rule should apply to those domain crossings that drive or connect to the specified pins, and also meet the requirements of the `-from` option.
  - If you combine `-to` and `-pins` options, the rule should apply to those domain crossings that are driven by or connected to the specified pins, and also meet the requirements of the `-to` option.
  - If you combine `-from`, `-to` and `-pins` options, the rule should apply to those domain crossings that
    - Are driven by or connected to the specified pins
    - Only drive logic in the specified *to* domains
    - Are driven by logic in the specified *from* domains
  - If you combine the `-pins` opttion with the `-force` option and with the`-from`, `-to`, or `-from` and `-to` option, the rule applies to all specified pins and the `-from` and `-to` options will be ignored.

Important

If you use the `-pins` option to select the domain crossings to be isolated, you must always combine this option with the `-force`, `-from`, `-to` or `-from` and `-to` options.

Options and Arguments

`-exclude` *`pin_list`*

Specifies that those *already selected* domain crossings that are connected to, driven by, or driving the specified pins must be excluded from isolation.

You can specify ports and instance pins. If the specified port or pin is not connected to a net segment selected with the `-from`, `-to` and `-pins` options, it will be ignored.

`-force`

Specifies that isolation logic shall be inserted, even in situations in which the rule would normally be ignored (see When Can Implementation Tools Ignore An Isolation Rule on page 99).

Note: If the isolation logic was already inserted, the tool will not insert the isolation logic again.

Note: You must specify this option together with the `-pins` option.

`-from` *`power_domain_list`*

Specifies that the rule must be applied to those domain crossings driven by logic in the specified (from) domain and with at least one leaf load in another power domain.

The power domain must have been previously defined with the `create_power_domain` command.

`-isolation_condition` *`expression`*

Specifies the condition when the selected domain crossings should be isolated. The condition can be a Boolean function of pins and ports.

If neither this option or the `-no_condition` option is specified, the rule is considered *incomplete*. In this case, only domain crossings connected to the primary input ports of the current design or macro model can be selected.

To complete an incomplete isolation rule, the `-default_isolation_condition` option specified with the `create_power_domain` command for the originating (driving) power domain(s) of the selected domain crossings will be used as the isolation condition.

If the `-default_isolation_condition` option for the driving power domain of the selected domain crossings was not specified, the incomplete rule is treated as a design constraint.

`-isolation_control` *`list_of_additional_controls`*

Specifies a list of controls.

Use the following format to specify a control:

`{` *`control_type expr`* `}`

`control_type` indicates the type of isolation control. Allowed types are:

- `sync_enable`—can only be used with isolation types `high` and `low`. The corresponding expression needs to be `true` before the isolation condition can be asserted and deasserted. But the expression can be corrupted when the isolation condition is asserted.
- `set`—can only be used with `-isolation_output hold`. When the expression is `true`, the stored value and isolation output will be set to 1 regardless of the stored state.
- `reset`—can only be used with `-isolation_output hold`. When the expression is `true`, the stored value and isolation output will be set to 0 regardless of the stored state.

`expr` is a simple boolean expression to describe the driving function of the control logic.

It is an error if the isolation condition becomes `true` when `sync_enable` is `false`.

It is an error to specify both `sync_enable` and either `set` or `reset` types of isolation controls at the same time.

To implement a rule with a `sync_enable` type of control, an isolation cell definition must exist in which the `-aux_enables` option specifies only one pin, and that pin must relate to the switchable supply.

`-isolation_output {low|high|hold|tristate|clamp_high|clamp_low}`

Controls the output value at the output of the isolation gates when the isolation condition is `true`.

The output can be `high`, `low`, in the `tristate`, held to the value it had right before the isolation condition is activated, or clamped to a high or low value.

A `tristate` output is implemented using a tristate buffer whose enable signal is determined by the isolation enable condition.

Rules defined with the `clamp_high` or `clamp_low` value for the isolation output, must be implemented with isolation cells that are defined with the `-clamp` option. In addition, the isolation target must be `from`.

Note: The `clamp_high` value can only be specified if the driving domain is ground switched. The `clamp_low` value can only be specified if the driving domain is power switched.

Default: `low`

`-isolation_target {from | to}`

Specifies when this rule applies.

- `from` indicates that the rule applies when the power domain of the *drivers* of the specified pins is switched off.
- `to` indicates that the rule applies when the power domain of the *loads* of the specified pins is switched off.

Default: `from`

Note: If two isolation rules apply to the same domain crossing and differ in isolation target, two isolation cells can be inserted.

Tip

If you intend to use the isolation rule to isolate cells with open source input pins or to isolate power clamp cells, the isolation target must be `to`.

`-name` *`string`*

Specifies the name of the isolation rule.

Note: The specified string cannot contain wildcards nor the hierarchy delimiter character.

`-no_condition`
: Specifies that the isolation logic is automatically enabled when the power domains containing the drivers of the selected domain crossings are shut off.

  Note: To implement this type of rule, only isolation cells defined with the `-no_enable` option can be used.

`-pins` *`pin_list`*
: Specifies to apply the rule to those domain crossings that are connected to, driven by, or driving the specified pins.

  You can specify ports and instance pins.

`-secondary_domain` *`domain`*
: Specifies the domain that provides the power supply for the isolation logic inferred by this rule.

  If the isolation logic is implemented with an isolation cell with secondary power or ground pins, this domain determines the supplies to which the secondary power and ground pins of the cell must be connected.

`-to` *`power_domain_list`*
: Specifies that the rule must be applied to those domain crossings that are connected to logic belonging to the specified (to) domain and that are driven by a signal from another power domain.

  The power domains must have been previously defined with the `create_power_domain` command.

Example

The following command creates isolation rule `iso2`, that applies to all domain crossings that are driven by or connected to pins whose names start with `DA` and `DB`, and that belong to domain `domainA`. However the rule does not apply if the domain crossings are driven or connected by pins `DA_Iso` and `DB_Iso` of domain `domainA`.

```
create_isolation_rule -name iso2 -pins { DA* DB* } -exclude { DA_Iso DB_Iso }\
    -from domainA
```

Related Information

Isolation Cell on page 19

create_power_domain on page 168

update_isolation_rules on page 255

create_level_shifter_rule

```
create_level_shifter_rule
     -name string
     { -force -pins pin_list
     | -from power_domain_list | -to power_domain_list
     | -from power_domain_list -to power_domain_list }
     [-pins pin_list]
     [-exclude pin_list][-bypass_condition expression]
     [-input_domain power_domain] [-output_domain power_domain]
```

Defines a rule for adding level shifters. A specific rule allows you to indicate on which domain crossings to insert level shifters. You can

- Select all domain crossings driven by logic in the domains specified with the `-from` option *and* that are driving logic in other power domains
- Select all domain crossings driving logic in the domains specified with the `-to` option *and* that are driven by logic from another power domain.
- Combine options to filter the set of domain crossings:
  - If you combine `-to` and `-from` options, the rule should apply to those domain crossings that only drive logic in the specified *to* domains and that are driven by logic in the specified *from* domains.
  - If you combine `-from` and `-pins` options, the rule should apply to those domain crossings that drive or connect to the specified pins, and also meet the requirements of the `-from` option.
  - If you combine `-to` and `-pins` options, the rule should apply to those domain crossings that are driven by or connected to the specified pins, and also meet the requirements of the `-to` option.
  - If you combine `-from`, `-to` and `-pins` options, the rule should apply to those domain crossings that
    - Are driven by or connected to the specified pins
    - Only drive logic in the specified *to* domains
    - Are driven by logic in the specified *from* domains
  - If you combine the `-pins` opttion with the `-force` option, and with the`-from`, `-to`, or -`from` and `-to` option, the rule applies to all specified pins and the -`from` and `-to` options will be ignored.

Important

If you use the `-pins` option to select the domain crossings for level shifter insertion, you must always combine this option with the `-force`, `-from`, `-to` or `-from` and `-to` options.

Tip

The `-input_domain` and `-output_domain` options can be used to connect the power and/or ground pin of a level shifter to the non-default power and/or ground net of the power domain in which the level shifter is instantiated.

Options and Arguments

`-bypass_condition` *`expression`*

Specifies the condition when to bypass the voltage shifting functionality.

When the expression evaluates to `true`, the logic associated with the rule does not perform voltage level shifting (is bypassed).

Note: To implement this rule, the tool must insert a bypass level shifter that was specified with the `-bypass_enable` option of the `define_level_shifter_cell` command.

`-exclude` *`pin_list`*

Specifies that those *already selected* domain crossings that are connected to, driven by, or driving the specified pins must be excluded from level shifter insertion.

You can specify ports and instance pins. If the specified port or pin is not connected to a net segment selected with the `-from`, `-to` and `-pins` options, it will be ignored.

`-force`

Specifies that a level shifter must be inserted, even when the rule would normally be ignored (see When Can Implementation Tools Ignore A Level Shifter Rule on page 94).

Note: If the level shifter logic was already inserted, the tool will not insert the level shifter logic again.

Note: You must specify this option together with the `-pins` option.

`-from` *power_domain_list*

Specifies to apply the rule to those domain crossings driven by logic in the specified (from) domain and with at least one leaf load in another power domain.

The power domain must have been previously defined with the `create_power_domain` command.

`-input_domain` *power_domain*

Specifies the input power domain of the level shifter inferred by this rule.

By default, the input power domain of a level shifter is the power domain of the logic that drives the net selected by this rule.

The input power and/or ground pin of the level shifter cell must be connected to the primary power and/or ground net of the input power domain of the level shifter.

Note: For the case of high to low level shifting, the level shifter cell may have no dedicated input power and ground pins. In this case, the input power domain specification will be ignored.

`-name` *string*

Specifies the name of the level shifter rule.

Note: The specified string cannot contain wildcards nor the hierarchy delimiter character.

`-output_domain` *power_domain*

Specifies the output power domain of the level shifter inferred by this rule.

By default, the output power domain of a level shifter is the power domain of the logic driven by the level shifter.

The output power and/or ground pin of the level shifter cell must be connected to the primary power and/or ground net of the output power domain of the level shifter.

Note: This option is required when an inferred level shifter drives logic that belongs to multiple power domains with different supplies.

`-pins` *pin_list*

Specifies to apply the rule to those domain crossings that are connected to, driven by, or driving the specified pins.

You can specify ports and instance pins.

`-to` *power_domain_list*

Specifies to apply the rule to those domain crossings that are connected to logic belonging to the specified (to) domain and that are driven by a signal from another power domain.

The power domain must have been previously defined with the `create_power_domain` command.

Example

The following command creates level shifter rule `lsr1`, that applies to all domain crossings that are driven by or connected to pins whose names start with `DA` and `DB`, and that belong to domain `PD1` or `PD2`. However the rule does not apply if the domain crossings are driven or connected by pin `DA_C` of domain `PD1` or `PD2`.

```
create_level_shifter_rule -name lsr1 -pins { DA* DB* } -exclude { DA_C }\
    -from PD1 -to PD2
```

Related Information

Level Shifter Cell on page 19

create_power_domain on page 168

define_level_shifter_cell on page 293

identify_power_logic on page 205

update_level_shifter_rules on page 260

create_mode

```
create_mode
     -name string -condition expression
     [-probability float] [-illegal]
```

Defines a mode of the design. Modes can be non-mutually exclusive, so multiple modes can be active simultaneously at a point in time. In addition, a mode definition does not require the explicit specification of the states of all power domains. As a result, this is a more general specification than power modes.

Options and Arguments

`-condition` *`expression`*

Specifies the condition when the design is in the specified mode. It can be described by a Boolean or arithmetic expression of the following objects:

- Design pins and ports
- Domain conditions in the following form:

  `power_domain@nominal_condition`
- Other mode or power mode definitions in the form of `@`*`mode`*

`-illegal`

Identifies the mode as illegal. By default, a mode is legal.

`-name` *`string`*

Specifies the name of the mode.

The name of a mode must be unique among other modes and power modes within the same design scope.

`-probability` *`float`*

Specifies a floating point value between 0 and 1 to indicate the probability of the design being in this mode.

Typically, the information is set by system architects and can be used by system level optimization tools to achieve power optimization by utilizing different mode configurations.

Example

In the following example, the design is in mode

- `M1`, when `PD3` is at nominal condition `nom1`

```
create_mode -name M1 -condition { PD3@nom1 }
```

- M2, when either
  - a&b is true
  - PD1 is at nom1, PD2 is at nom2, and the design is in mode M1

```
create_mode -name M2 -condition { (a&b) | ((PD1@nom1 & PD2@nom2) & @M1) }
```

Related Information

Mode on page 16

create_power_mode on page 177

create_mode_transition

```
create_mode_transition
     -name string
     -from power_mode -to power_mode
     [-assertions assertion_list]
     { -start_condition expression [-end_condition expression]
       [ -cycles [integer:]integer -clock_pin clock_pin
       | -latency [float:]float ]
     | -illegal }
```

Describes how the transition between two modes is controlled, and the time it takes for each power domain to complete the transition. The transition can be between either power modes or generic modes.

To determine the starting time of a specific mode transition for a power domain, use the following process:

1. If the power domain is shut off at the end of a mode transition, the domain starts the transition when the shutoff condition changes to `true`.
2. If the nominal condition of the ending state is specified in the `-active_state_conditions` of the power domain, the domain starts the transition to the ending state when the condition for this nominal condition changes to `true`
3. If the power domain is shut off at the beginning of a mode transition, the domain starts the transition when the shutoff condition changes to `false`.
4. For all other cases, the domain starts the transition as soon as the expression specified in `-start_condition` becomes `true`.

Tip

Mode transitions that start from the same mode cannot have the same start condition.

Options and Arguments

`-assertions` *`assertion_list`*

Selects only the assertions whose names are included in the assertion list. The assertions must evaluate to `true` for the specified mode transition to take place.

Assertion names can contain wildcards.

`-clock_pin` *clock_pin*

Specifies the name of the clock pin that controls the transition.

You can specify ports and instance pins.

`-cycles [`*integer*`:]`*integer*

Specifies an integer of number of clock cycles needed to complete the power mode transition.

If two numbers are specified, the first number indicates the minimum number of clock cycles, while the second number indicates the maximum number of clock cycles.

Note: If you specify only one value, it is considered to be the maximum number.

`-end_condition` *expression*

Specifies the condition that acknowledges that the design is in the power mode specified by the `-to` option.

The expression is a Boolean function of pins and ports.

`-from` *power_mode* Specifies the power mode from which to transit.

The mode must have been previously defined with the `create_mode` or `create_power_mode`.

`-illegal` Identifies the power mode transition as an illegal one.

`-latency [`*float*`:]`*float*

Specifies the time needed to complete the power mode transition. Specify the time in the units specified by the `set_time_unit` command.

If two numbers are specified, the first number indicates the minimum time needed, while the second number indicates the maximum time needed.

Note: If you specify only one value, it is considered to be the maximum number.

`-name` *string* Specifies the name of the power mode transition.

Note: The specified string cannot contain wildcards nor the hierarchy delimiter character.

`-start_condition` *`expression`*

Specifies the condition that triggers the power mode transition.

`-to` *`power_mode`*

Specifies the power mode to which to transit.

The mode must have been previously defined with the `create_mode` or `create_power_mode`.

Related Information

Mode Transition on page 16

create_mode on page 154

create_power_mode on page 177

create_nominal_condition

```
create_nominal_condition
     -name string
     -voltage {voltage | voltage_list}
     [-ground_voltage {voltage | voltage_list}]
     [-state {on | off | standby}]
     [-pmos_bias_voltage {voltage | voltage_list}]
     [-nmos_bias_voltage {voltage | voltage_list}]
     [-deep_pwell_voltage {voltage | voltage_list}]
     [-deep_nwell_voltage {voltage | voltage_list}]
```

Creates a nominal operating condition with the specified voltage. For each voltage, you can specify a single value, two values, or three values:

- If you specify one value, you must specify the nominal voltage. For example,

  ```
  -voltage 0.9
  ```

- If you specify two values, specify the minimum and maximum voltages. The nominal voltage is considered to be the average of the minimum and maximum voltages. For example,

  ```
  -voltage { 0.7 1.1 }
  ```

- If you specify three values, you must specify the voltages in the following order: minimum, nominal, and maximum. For example,

  ```
  -voltage { 0.7 0.9 1.1 }
  ```

- If you specify a voltage list, valid values include the minimum, but not the maximum value:

  $$min \leq voltage < max$$

Note: A power domain is switched off if the voltage of its associated nominal condition is 0.

Options and Arguments

`-deep_nwell_voltage {voltage | voltage_list}`

Specifies either a single voltage or a voltage list for the deep nwell supply net in the domain that uses this condition.

The voltage must be specified in volts (V).

`-deep_pwell_voltage {voltage | voltage_list}`

Specifies either a single voltage or a voltage list for the deep pwell supply net in the domain that uses this condition.

The voltage must be specified in volts (V).

`-ground_voltage {`*`voltage | voltage_list`*`}`

Specifies either a single voltage or a voltage list for the ground net in the domain that uses this condition.

The voltage must be specified in volts (V).

`-nmos_bias_voltage {`*`voltage | voltage_list`*`}`

Specifies either a single voltage or voltage list for the body bias voltage of the n-type transistors in the domain that uses this condition.

The voltage must be specified in volt (V).

`-name` *`string`* Specifies the name of the nominal operating condition.

Note: The specified string cannot contain wildcards nor the hierarchy delimiter character.

`-pmos_bias_voltage` *`float`*

Specifies either a single voltage or voltage list for the body bias voltage of the p-type transistors in the domain that uses this condition.

The voltage must be specified in volt (V).

`-state {on | off | standby}`

Specifies the state of a power domain when it uses this nominal condition.

Note: In the description below, *logic* refers to logic in the associated power domain that does not depend on a secondary domain.

You can specify the following values:

- `on`—indicates that all logic in the domain operates at operational speed. To improve performance in the on state, you can apply forward body biasing.

  To describe forward body biasing, you must specify `-pmos_bias_voltage` with a value smaller than the value of `-voltage` and `-nmos_bias_voltage` with a value greater than the value of `-ground_voltage`.

- `off`—indicates that all logic in this power domain is shut down and does not retain its state.

- `standby`—indicates that all logic in the domain retain their state values but that the logic cannot perform any normal operation. The standby state can be achieved by

  - reverse body biasing

    To describe reverse body biasing, you must specify `-pmos_bias_voltage` with a value larger than the value of `-voltage` and `-nmos_bias_voltage` with a value smaller than the value of `-ground_voltage`.

  - source biasing

    To describe source biasing, you must specify the voltage values for the power supply (-voltage) and ground supply (`-ground_voltage`) that cause the logic to be in the standby state.

By default, the state is considered on if the voltage specified for the `-voltage` option is non-zero.

`-voltage {`*`voltage`* | *`voltage_list`*`}`

Specifies either a single voltage or a voltage list for the power net in the domain that uses this condition.

The voltage must be specified in volt (V).

Example

The following example applies to a design with three power domains (PD1, PD2, and PD3). The design can operate in three modes (M1, M2, and M3).

This example shows that you can create an explicit nominal condition for a domain that is switched off.

The following table shows the voltages for each power domain in each of the modes.

Power Mode	Power Domain		
	PD1	**PD2**	**PD3**
M1	1.2V	1.2V	1.0V
M2	1.2V	1.0V	0.0V
M3	1.0V	0.0V	0.0V

```
create_nominal_condition -name high -voltage 1.2
create_nominal_condition -name low -voltage 1.0
create_nominal_condition -name off -voltage 0
create_power_mode -name M1 -domain_conditions {PD1@high PD2@high PD3@low}
create_power_mode -name M2 -domain_conditions {PD1@high PD2@low PD3@off}
create_power_mode -name M3 -domain_conditions {PD1@low PD2@off PD3@off}
```

Related Information

Nominal Operating Condition on page 16

create_power_mode on page 177

update_nominal_condition on page 265

create_operating_corner

```
create_operating_corner
     -name corner
     -voltage float [-ground_voltage float]
     [-pmos_bias_voltage float] [-nmos_bias_voltage float]
     [-process float]
     [-temperature float]
     -library_set library_set_list [-power_library_set library_set_list]
```

Defines an operating corner and associates it with a library set.

Note: The voltage specified in the corner definition associated with a power domain in an analysis view must be consistent with the voltage specified in the nominal condition associated with this domain in the corresponding power mode of the analysis view.

Options and Arguments

`-ground_voltage` *float*

Specifies the ground supply voltage in volt (V).

`-library_set` *library_set_list*

References one or more library sets, where each library set is created for a specific PVT. Tools should automatically select the correct timing/power models to be used for the PVT specified by this command.

Each library set must have been previously defined with the `define_library_set` command.

Note: These library sets are used for timing analysis and optimization. When the `-power_library_set` option is not specified, these library sets are also used for power analysis and optimization.

`-name` *corner*

Specifies the name of the operating corner you want to create.

Note: The specified string cannot contain wildcards nor the hierarchy delimiter character.

`-nmos_bias_voltage` *float*

Specifies the body bias voltage of the n-type transistors in the domain that uses this corner.

The voltage must be specified in volt (V).

`-pmos_bias_voltage` *`float`*

Specifies the body bias voltage of the p-type transistors in the domain that uses this corner.

The voltage must be specified in volt (V).

`-power_library_set` *`library_set_list`*

Specifies one or more library sets for power analysis and optimization.

The library sets must have been previously defined with the `define_library_set` command.

Note: If this option is not specified, the timing libraries specified with the `-library_set` option will be used for power analysis and optimization.

`-process` *`float`* Specifies the process value of the corner. This value depends on the used technology process and is provided by the library vendor.

If this option is not specified, the value defaults to the value specified in the first library of the specified library set.

`-temperature` *`float`* Specifies the temperature of the operating condition in degrees Celsius.

If this option is not specified, the value defaults to the value specified in the first library of the specified library set.

`-voltage` *`float`* Specifies the voltage of the operating condition in volt.

Example

The following example uses two library sets to create an operating corner.

```
define_library_set -name set1 -libraries {a1.lib b1.lib}
define_library_set -name set2 -libraries {a2.lib b2.lib}
create_operating_corner -name ... -voltage ... -library_set {set1 set2}
```

Related Information

Operating Corner on page 16

define_library_set on page 194

create_pad_rule

```
create_pad_rule -name string
     {-of_bond_ports port_list | -instances instance_list}
     -mapping {mapping_list}
```

Defines how to map pin groups or power domains of pad instances to top-level power domains.

Options and Arguments

`-instances` *`instance_list`*

Specifies a list of instances of pad cells.

A selected instance can be defined as a pad cell or as a CPF macro model.

`-mapping` *`mapping_list`*

Specifies the mapping of each pin group of the pad cell definition or a power domain in macro model definition to a top-level domain.

Use the following format to specify a mapping:

`{group_id_or_macro_model_domain power_domain}`

where

`group_id_or_macro_model_domain` references

- A pin group defined for the pad cell with the `define_pad_cell` command
- A power domain when the pad cell is defined as CPF macro model

  If a cell is defined both as a pad cell and macro model, the macro model takes precedence .

`power_domain` is the name of a top-level power domain.

`-name` *`string`*

Specifies the name of the pad rule.

Note: The specified string cannot contain wildcards nor the hierarchy delimiter character.

`-of_bond_ports` *`port_list`*

Selects pad cell instances that are connected to the specified list of top-level bonding ports.

A selected instance can be defined as a pad cell or as a CPF macro model.

Example

The following command defines cell `VDDC_STGIN` as a pad cell. It defines three pins in the `CVDD` pin group. The pins in this group are in the definition of pad rule `pad1` mapped to the `PD_CORE` top domain. The remaining pins of the pad cell belong to the `DEFAULT` group.

```
define_pad_cells -cells VDDC_STGIN \
-pad_pins pad -pin_groups { CVDD:{ VDDCORE VDDC2CORE pad} }
...
create_pad_rule  -name pad1  -of_bond_ports VDDC \
-mapping { {CVDD PD_CORE} {DEFAULT PD_PAD} }
```

Related Information

define_pad_cell on page 303

Modeling a Pad Cell in the *Common Power Format User Guide*

create_power_domain

```
create_power_domain
     -name power_domain
     [-instances instance_list] [-exclude_instances instance_list]
     [-boundary_ports pin_list [-exclude_ports pin_list]] [-default]
     [-shutoff_condition expression [-external_controlled_shutoff]]
     [-default_isolation_condition expression ]
     [-default_restore_edge expr | -default_save_edge expr
     |-default_restore_edge expr -default_save_edge expr
     |-default_restore_level expr -default_save_level expr ]
     [-power_up_states {random|high|low|inverted} ]
     [-power_down_states {low|high|random|inverted}]
     [-active_state_conditions active_state_condition_list ]
     [-base_domains domain_list] [-power_source]
```

Creates a power domain and specifies the instances and boundary ports and pins that belong to this power domain.

A *pure virtual power domain* is a power domain that is not specified as the default power domain, and for which no instances or boundary ports are defined.

By default, an instance inherits the power domain setting from its parent hierarchical instance or the design, unless that instance was associated with a specific power domain. In addition, all top-level boundary ports are considered to belong to the default power domain, unless they have been associated with a specific domain.

In CPF, power domains are associated with the design objects based on the order of the logical hierarchy. The order in which you create the power domains is irrelevant.

You must define at least one power domain for a design, and one (and only one) power domain must be specified as the default power domain. However, for macro models, power domains must only be specified if the macro models contain ***any*** logic. If the macro model consist of only power and ground nets, floating ports, and `set_wire_feedthrough` nets, a domain is not required.

The top design, identified by the first `set_design` command, belongs to the default power domain.

Important

Power domains are scope-specific. If you change the scope to a hierarchical instance (using the `set_instance` command), a power domain definition can only apply to that hierarchical instance or to the instances (children) of that hierarchical instance. This rule also applies to the default power domain defined in this scope.

Options and Arguments

`-active_state_conditions` *`active_state_condition_list`*

Specifies the Boolean condition for each nominal condition or state at which the power domain is considered on or in standby state.

When a condition changes to true, the power domain starts the transition to the state specified by the nominal condition.

Use the following format to specify an active state condition:

`nominal_condition_name@expression`

The *`nominal_condition_name`* must have been specified with a `create_nominal_condition` command.

The expression is a regular CPF expression. The expression must be enclosed in braces.

List all nominal conditions that appear with the specified domain in a domain condition for any of the defined power modes.

Note: None of the conditions can overlap and the domain must be either in one of the active states or be shut off.

Note: This option is only needed if the power domain can operate on different supply voltages and its operation is controlled by a set of signals.

`-base_domains` *`domain_list`*

Specifies a list of base domains that supply external power and/or ground to the primary domain through some switch network.

When all base power domains are switched off, the power domain will be switched off irrespective of the shutoff conditions.

If one of the base domains is on or in standby state and the shutoff condition is `false`, the power domain is considered on or in standby state.

`-boundary_ports` *`pin_list`*

Specifies the list of *data* inputs and outputs that are considered part of this domain.

- For inputs and outputs of the top-level design, specify ports.

  When you assign a port of a design to a power domain, it is implied that the logic outside the design that is connected to this port is powered by the power supply of this domain.

- For inputs, outputs, and inouts of macro cell instances, specify a list of the instance pins that are part of the domain.

  When you assign a boundary port of a macro cell to a power domain, it is implied that the logic inside the macro cell connected to this port is powered by the power supply of this domain.

  Note: Inout ports specified as boundary ports are only used for verification. No isolation cell or level shifter will be inserted on an inout port.

Note: Hierarchical instance pins will be ignored if specified.

Note: In case of a macro model, the following applies:

- A boundary input port driving logic in multiple power domains can be associated with all those power domains.
- All boundary pins will be specified in the macro model domain definition. As a result, you do not need to specify the boundary pins in the upper-level power domain. It is an error if the upper-level domain definition and the macro model domain definition have conflicting power domain specifications.
- Power and ground ports of a macro model will be ignored when specified. They must be properly assigned to the power domain using the `-primary_power_net` and `-primary_ground_net` options of the `update_power_domain` command.

`-default`

Identifies the specified domain as the default power domain.

All instances of the design that were *not* associated with a specific power domain belong to the default power domain.

`-default_isolation_condition` *`expression`*

Specifies the default isolation condition for isolation rules without isolation condition that apply to net segments with leaf driver in this domain.

The expression is a Boolean function of pins. When the expression changes to `true`, the isolation logic is enabled.

`-default_restore_edge` *`expr`* (`-default_restore_level` *`expr`*)

Specifies the default condition when the states of the sequential elements need to be restored for all state retention rules created for sequential instances in this power domain.

If no state retention rules were created for this power domain or a list of sequential elements in this domain, this option is ignored.

The expression (*`expr`*) can be a Boolean function of pins and ports.

- If the condition is edge-sensitive, the states are restored when the expression *changes* from `false` to `true`.
- If the condition is level-sensitive, the states are restored as long as the expression is `true`.

`-default_save_edge` *`expr`* (`-default_save_level` *`expr`*)

Specifies the default condition when the states of the sequential elements need to be saved for all state retention rules created for sequential instances in this power domain.

If no state retention rules were created for this power domain, this option is ignored.

The expression (*`expr`*) can be a Boolean function of pins and ports.

- If the condition is edge-sensitive, the states are saved when the expression *changes* from `false` to `true`.
- If the condition is level-sensitive, the states are saved as long as the expression is `true`.

`-exclude_instances` *`instance_list`*

Specifies to exclude those *instances already selected* through the `-instances` option.

If the specified instance is not selected through the `-instances` option, it will be ignored.

`-exclude_ports` *`pin_list`*

Specifies to exclude those *pins already selected* through the `-boundary_ports` option.

If the specified pin is not selected through the `-boundary_ports` option, it will be ignored.

`-external_controlled_shutoff`

Specifies that the power domain being defined is an external switchable power domain.

`-instances` *`instance_list`*

Specifies the names of all instances that belong to the specified power domain.

If this option is specified together with the `-boundary_ports` option, it indicates that for any connection between a specified port and any instance inside the power domain, no special interface logic for power management is required.

Instances referred to in a `create_power_domain` between a `set_design` and `end_design` pair can be hierarchical instances or instances of any library cells, including standard cells and macro cells.

Instances referred to in a `create_power_domain` between a `set_macro_model` and `end_macro_model` pair can be registers in the behavioral model of the macro cell.

`-name` *`power_domain`*

Specifies the name of a power domain.

Note: The specified string cannot contain wildcards nor the hierarchy delimiter character.

`-power_down_states {low|high|random|inverted}`

Specifies the value with which to overwrite the default corrupt semantics at the outputs of instances in a power domain after the domain is switched off.

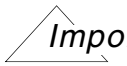
- `high`—the output values are set to1
- `low`—the output values are set to 0
- `random`—the output values are randomly set to 0 or 1
- `inverted`—the output values are set to the inversion of the values before the power domain is switched off

Default: `low`.

When this option is ommitted, simulation tools should use the default power down semantics for simulation.

`-power_source`

Indicates that the power domain is a power source domain, that is, it models a supply from an on-chip regulator, such as a Low Dropout (LDO) regulator.



This option can only be specified for a power domain within a macro model.

See also Modeling a Voltage Regulator.

`-power_up_states {random | high | low | inverted}}`

Specifies the state to which the non-state-retention cells in this power domain must be initialized after powering up the power domain.

- `high`: all non state-retention registers are initialized to1 after power-up
- `low`: all non state-retention registers are initialized to 0 after power-up
- `random`: all non state-retention registers are randomly initialized to 0 or 1 after power-up
- `inverted`—all non state-retention registers are initialized to the inversion of the values before the power domain is switched off

Default: `random`.

If the option is ommitted, simulation tools should not initialize the states after powering up.

`-shutoff_condition` *`expression`*

Specifies the condition when a power domain is shut off. The condition is a Boolean function of pins and ports.

Examples

- For the following example, assume a design with the following hierarchy:

```
Top
    INST1
        INST2
```

The following two sets of CPF commands are equivalent:

a.
```
create_power_domain -name PD1 -instances INST1
create_power_domain -name PD2 -instances INST1.INST2
```

b.
```
create_power_domain -name PD2 -instances INST1.INST2
create_power_domain -name PD1 -instances INST1
```

This illustrates that the order in which you specify the target domains is irrelevant. The result is that instance `INST1` belongs to power domain `PD1` and instance `INST2` belongs to power domain `PD2`.

- The following command associates a list of instances with power domain `PD2`.

```
create_power_domain -name PD2 -instances {A*C I_ARM1 PAD1}
```

- The following command defines the active state conditions for power domain `foo`.

```
create_power_domain -name foo -instances {...} -shutoff_condition !pso
 -active_state_conditions { 1.0v@vdd1_en 1.2v@{!vdd1_en}}
```

- The following example illustrates that boundary input ports of a macro model can be associated with multiple domains. Input port M of macro `cellA` drives logic in domains `PD2` and `PD3`. Therefor, port M is listed as boundary port of both domains.

```
set_macro_model CellA

create_power_domain -name PD1 -default -boundary_ports { psw[0]}
create_power_domain -name PD2 -boundary_ports {M X2 Y2}
create_power_domain -name PD3 -boundary_ports {M X3} ...
create_power_domain -name PD4 -boundary_ports {N YN} ....

update_power_domain -name PD1 ...
update_power_domain -name PD2 ....
....
end_macro_model
```

- Assume the top level has the following instances: B, A1, A2, A1/B, A2/B. In the following command, `-instances` initially selects top-level instances `A1` and `A2` as members of domain `PD1` but because of the `-exclude` option, `A2` will not be included as a member for `PD1`. `A1/B` is ignored because `A1/B` was not selected through the `-instances` option.

```
create_power_domain -name PD1 -instances A* -exclude {A2 A1/B} ...
```

- The following example declares power domain `PDVOUT` as a power source domain.

```
set_macro_model regulator
create_power_domain -name PDVIN
update_power_domain -name PDVIN -primary_power_net HAVDD -primary_ground_net AVSS
create_power_domain -name PDVOUT -default -base_domains {PDVIN} -power_source
update_power_domain -name PDVOUT -pmos_bias_net VBB
...
create_nominal_condition -name LDO_range -voltage 1.1 \
-pmos_bias_voltage {1.1 1.3}
...
create_power_mode -name PM -default -domain_conditions \
    { PDREF@REF PDVIN@HVDD PDVOUT@LDO_range }
...
end_macro_model regulator
```

Note: The primary supply voltage of the base domain can be different from the primary supply voltage of the power source domain.

Related information

Power Domain on page 17

For more information on modeling different power domain categories, refer to Power Domain Categories on page 40.

Referencing CPF Objects on page 125

create_global_connection on page 140

create_isolation_rule on page 145

create_nominal_condition on page 159

create_power_switch_rule on page 184

create_state_retention_rule on page 186

update_power_domain on page 266

create_power_mode

```
create_power_mode
     -name string [-default]
     {-domain_conditions domain_condition_list
     |-group_modes group_mode_list
     |-domain_conditions domain_condition_list -group_modes group_mode_list }
     [-condition expression]
```

Defines a power mode.

If your design has more than one power domain, and you want to use the same CPF file for simulation, implementation, and verification purposes, you must define at least one power mode.

If you define any power mode, you must define one (and only one) power mode as the default mode within a scope.

A power mode that has any combination of power supplies that is not covered by a legal power mode is illegal.

Important

You must ensure that any voltage that is contained in the voltage range of several nominal conditions associated with the same power domain, has the same state value in all these nominal conditions.

Note: Illegal power modes should be treated as an error by verification tools, but may be used by implementation tools to permit optimization.

Options and Arguments

`-condition` *expression*

Specifies the condition when the design is in the specified power mode.

The expression is a Boolean function of pins and ports.

Note: This option applies only to verification and simulation tools.

`-default`

Labels the specified mode as the default mode. The default mode is the mode that corresponds to the initial state of the design.

Note: For a given power mode control group you can only have one power mode as default.

`-domain_conditions` *`domain_condition_list`*

Specifies the nominal condition of each power domain to be considered in the specified power mode.

Use the following format to specify a domain condition:

`domain_name@nominal_condition_name`

A domain is considered in an `off` state in the specified mode if

- It is associated with a nominal condition whose state is off
- It is not specified in the list of domain conditions

  In this case, the power supply voltage of the switched-off domain is considered to be 0.0V.

Note: You can associate each power domain with only one nominal condition for a given power mode.

The power domains must have been previously defined with the `create_power_domain` command.

The condition must have been previously defined with the `create_nominal_condition` command.

`-group_modes` *`group_mode_list`*

Specifies the mode of each power mode control group to be considered with the defined mode.

Use the following format to specify the mode for a group:

`group_name@power_mode_name`

The specified *group_name* refers to the name of a power mode control group, previously defined with the set_power_mode_control_group command or to the name of the scope in case the power control group was automatically defined. The specified *power_mode_name* must have been defined in a create_power_mode command contained in the definition of the specified power mode control group.

If a power mode control group in a lower scope is not referred to by another power mode or analysis view in an upper scope, the power mode or analysis view at the block level will be ignored at the top level. However, if a power mode control group in a lower scope is referred by some other power mode or analysis view in an upper scope, but not by all, the power mode control group is supposed to be in the default mode in those modes in the upper scope where it is not referenced.

`-name` *string*

Specifies the name of the mode.

This power mode *cannot* be defined with the assert_illegal_domain_configurations command.

Note: The specified string cannot contain wildcards nor the hierarchy delimiter character.

Examples

- The following example applies to a design with three power domains (PD1, PD2, and PD3). The design can operate in two modes (M1, and M2).

  The following table shows the voltages for each power domain in each of the modes.

Power Mode	Power Domain		
	PD1	**PD2**	**PD3**
M1	1.2V	1.2V	1.0V
M2	1.2V	1.0V	0.0V

```
create_nominal_condition -name high -voltage 1.2
create_nominal_condition -name low -voltage 1.0
```

```
create_power_mode -name M1 -domain_conditions {PD1@high PD2@high PD3@low}
create_power_mode -name M2 -domain_conditions {PD1@high PD2@low}
```

Note: Since power domain `PD3` is switched off in mode `M2`, you can omit it from the domain conditions when you define mode `M2`.

- The following example shows a design with four power domains that can all be switched independently. To specify all combinations of the power domain states, you would need 16 (2^4) power modes. However, you only need five (4+1) power modes to model the behavior of the design as shown in the table below:

Power Mode	Power Domains			
	PD1	**PD2**	**PD3**	**PD4**
M1	on	on	on	on
M2	off	on	on	on
M3	on	off	on	on
M4	on	on	off	on
M5	on	on	on	off

For example, consider the following combination:

PD1	**PD2**	**PD3**	**PD4**
off	off	on	on

Any `on` to `off` or `off` to `on` crossing is covered by either modes `M2` or `M3`. The crossing between domains `PD1` and `PD2` can be ignored because both domains are switched off in this case.

- In the following example, the nominal conditions that power domain `PD1` is associated with have an overlapping voltage ranges [`0.7:0.8`). This will cause an error because different state values are defined for these voltages.

```
create_nominal_condition -name nc1 -state on -voltage {0.7 1.0}
create_nominal_condition -name nc2 -state standby -voltage {0.5 0.8}
create_power_mode -name M1 -domain_conditions {PD1@nc1 ....}
```

```
create_power_mode -name M2 -domain_conditions {PD1@nc2 ....}
```

Related Information

Power Mode on page 18

Illegal Domain Configurations on page 55

Power Mode Control Groups on page 81

create_nominal_condition on page 159

create_power_domain on page 168

set_power_mode_control_group on page 237

update_power_mode on page 272

create_power_nets

```
create_power_nets
     -nets net_list
     [-voltage {float | voltage_range}]
     [-external_shutoff_condition expression | -internal]
     [-user_attributes string_list]
     [-peak_ir_drop_limit float]
     [-average_ir_drop_limit float]
```

Specifies or creates a list of power nets.

Note: Even if this net exists in the RTL or the netlist, it still must be declared through this command if the net is referenced in other CPF commands.

The power nets are created within the current scope.

Options and Arguments

`-average_ir_drop_limit` *float*

Specifies the maximum allowed average IR drop on the specified power nets due to resistive effects in volt (V) for any mode.

Default: `0`

`-external_shutoff_condition` *expression*

When the specified power nets are powered by an external power source, you can use an expression to specify under which condition the power source can be switched off.

In this case, the net must be connected to an I/O port or pad.

Note: If neither this option nor the `-internal` option is specified, the power source is assumed to be the primary source of an unswitched power domain.

`-internal`

Specifies that the nets must be driven by an on-chip power switch. You must specify this option if this power net is the primary power net of an internal switchable power domain using power (header) switches.

`-nets` *`net_list`*
: Declares a list of power nets.

  Note: You can specify a hierarchical net name to infer a power net in another scope.

`-peak_ir_drop_limit` *`float`*
: Specifies the maximum allowed peak IR drop on the specified power nets due to resistive effects in volt (V) for any mode.

  Default: `0`

`-user_attributes` *`string_list`*
: Attaches a list of user-defined attributes to the net. Specify a list of strings.

`-voltage {`*`float`* | *`voltage_range`*`}`
: Identifies the voltage applied to the specified nets.

  The voltage range must be specified as follows:

  `lower_bound:upper_bound`

  Specify the lower bound and upper bound, respectively.

Example

- The following command declares power nets `vdd3` and `vdd4` with voltage of `0.8`.

```
create_power_nets -voltage 0.8 -nets {vdd3 vdd4}
```

create_power_switch_rule

```
create_power_switch_rule
     -name string
     -domain power_domain
     {-external_power_net net | -external_ground_net net}
```

Specifies how a single on-chip power switch must connect the external power or ground nets to the primary power or ground nets of the specified power domain.

You can specify one or more commands for a power domain depending on whether you want to control the power domain by a single switch or multiple switches.

By default, the proper power switch cell will be selected from the cells specified through the `define_power_switch_cell` command. To use a specific cell, use the `update_power_switch_rule` command.

By default, the inversion of the expression specified for the shutoff condition of the power domain is used as the driver for the enable pin of the power switch cell. For complicated cells with multiple enable pins, or if you want to use a different signal to drive the enable pins, use the `update_power_switch_rule` command.

Options and Arguments

`-domain` *power_domain*

Specifies the name of a power domain.

The power domain must have been previously defined with the `create_power_domain` command.

`-external_ground_net` *net*

Specifies the external ground net to which the source pin of the ground switch must be connected. The drain pin must be connected to the primary ground net associated with the specified power domain.

You can only specify this option when you use a footer cell.

When you create a power switch rule for a macro cell, this option specifies the boundary port for the external supply of this ground switch in the macro cell. Otherwise, you must have declared this net using the `create_ground_nets` command.

Note: A net specified as an external ground net in one domain can be specified as an primary ground net of another domain using the `-primary_ground_net` option of the `update_power_domain` command.

`-external_power_net` *`net`*

Specifies the external power net to which the source pin of the power switch must be connected. The drain pin must be connected to the primary power net associated with the specified power domain.

You can only specify this option when you use a header cell.

When you create a power switch rule for a macro cell, this option specifies the boundary port for the external supply of this power switch in the macro cell. Otherwise, you must have declared this net using the `create_power_nets` command.

Note: A net specified as an external power net in one domain can be specified as an primary power net of another domain using the `-primary_power_net` option of the `update_power_domain` command.

`-name` *`string`*

Specifies the name of the power switch rule.

Note: The specified string cannot contain wildcards nor the hierarchy delimiter character.

Example

- In the following example, power domain `X` is made switchable through a single switch.

```
create_power_domain -name X -shutoff_condition !ena -instances ...
update_power_domain -name X -primary_power_net VDD_SW
create_power_switch_rule -name ps1 -domain X -external_power_net VDD
```

Related Information

Power Switch Cell on page 20

create_power_domain on page 168

define_power_switch_cell on page 308

update_power_domain on page 266

update_power_switch_rule on page 276

create_state_retention_rule

```
create_state_retention_rule
     -name string
     {-domain power_domain | -instances instance_list}
     [-exclude instance_list]
     [-required]
     [ -restore_edge expr | -save_edge expr
          |-restore_edge expr -save_edge expr
          |-restore_level expr -save_level expr]
     [-restore_precondition expr] [-save_precondition expr]
     [-retention_precondition expr]
     [-target_type {flop|latch|both}]
     [-secondary_domain domain] [-use_secondary_for_output]
```

Defines the rule for replacing selected registers or all registers in the specified power domain with state retention registers.

Note: To implement this rule, only cells defined with the define_state_retention_cell command can be used.

Options and Arguments

`-domain` *power_domain*

Specifies to apply the rule to all registers or non-state-retention sequential instances in this domain.

The rule is ignored if the specified power domain is always on or if the domain has no sequential elements.

The power domain must have been previously defined with the `create_power_domain` command.

`-exclude` *`instance_list`*

Specifies to exclude the specified list of instances from the list of selected instances that must be replaced with state retention elements.

An instance can be a

- Leaf or hiearchical instance name in a gate-level netlist
- Register variable or hiearchical instance in RTL

If you specify the name of a hierarchical instance, all registers or non-state-retention sequential elements in this instance and its children will be excluded.

`-instances` *`instance_list`*

Specifies the instances that you want to replace with a state retention register.

An instance can be a

- Leaf or hiearchical instance name in a gate-level netlist
- Register variable or hiearchical instance in RTL

If you specify the name of a hierarchical instance, all registers or non-state-retention sequential elements in this instance and its children will be replaced.

Note: The specified instances can belong to several power domains. If they belong to different power domains, the same conditions will be applied to all of them.

`-name` *`string`*

Specifies the name of the state retention rule.

Note: The specified string cannot contain wildcards nor the hierarchy delimiter character.

`-required`

Indicates that the registers selected by this rule must be implemented using retention flops or latches.

If you omit this option, the selected registers may or may not be implemented as retention logic, depending on the top-level rule specification.

When any of the control conditions `-save_level`, `-restore_level`, `-save_edge`, or `-restore_edge` is specified, the rule must be implemented whether or not the `-required` option is specified.

It is an error if a register is selected by an incomplete retention rule (see State Retention Rules on page 107) with the `-required` option specified, and either

- No other complete retention rule would select this register
- No default condition is specified for the domain that owns the register: for example, `-default_save_edge` in the `create_power_domain` command

`-restore_edge` *`expr`*

Specifies that the states are restored when the expression *changes* from `false` to `true`.

The expression (*`expr`*) can be a Boolean function of the pin or port that directly controls the restore operation.

If you specify this option with the `create_state_retention_rule` and the `-default_restore_edge` option with the `create_power_domain` command, the option specified with this command takes precedence.

If this option is omitted but the `-save_edge` option is specified, the registers restore the saved values when the power is turned on.

If you omitted both the `-restore_edge` option and the `-save_edge` option, but you specified the `-default_restore_edge` option with the `create_power_domain` command for the corresponding power domain(s), then that condition will be used.

If this option is specified, the rule must be implemented whether or not the `-required` option is specified.

`-restore_level` *`expr`*

Specifies that the states are restored when the restore expression is `true` and the power is on.

The expression (*`expr`*) can be a Boolean function of the pin or port that directly controls the restore operation.

If you specify this option with the `create_state_retention_rule` and the `-default_restore_level` option with the `create_power_domain` command, the option specified with this command takes precedence.

If you omitted both the `-restore_level` option and the `-save_level` option, but you specified the `-default_restore_level` and `-default_save_level` options with the `create_power_domain` command for the corresponding power domain(s), then these conditions will be used.

If this option is specified, the rule must be implemented whether or not the `-required` option is specified.

`-restore_precondition` *`expr`*

Specifies an additional condition that must be met for the restore operation to be successful. The condition can be related to the clock, set and reset signals, but should not include signals that directly control the save and restore function of the cell.

If this option is not specified, the restore operation is performed when the `restore_edge` (`restore_level`) condition evaluates to `true`.

The operands of the expression (*`expr`*) can be pins, ports or nets. The pins or ports that control the save or restore operation should not be included in this expression.

This option is only intended for RTL simulation.

`-retention_precondition` *expr*

Specifies an additional condition that must be met (between the time that the power domain of the retention logic is powered down and powered up) for the retention operation to be successful. The condition can be related to the clock, set and reset signals, but should not include signals that directly control the save and restore function of the cell.

The expression (*expr*) can be a Boolean function of pins, ports and nets.

If the condition is violated, the retention operation will fail and the simulation tools will corrupt the saved values.

`-save_edge` *expr*

Specifies that the states are saved when the expression *changes* from `false` to `true` and the power is on.

The expression (*expr*) can be a Boolean function of the pin or port that directly controls the save operation.

If you specify this option with the `create_state_retention_rule` and the `-default_save_edge` option with the `create_power_domain` command, the option specified with this command takes precedence.

If this option is omitted but the `-restore_edge` option is specified, the states are saved when the expression changes from `true` to `false` and the power is on.

If you omitted both the `-restore_edge` option and the `-save_edge` option, but you specified the `-default_save_edge` option with the `create_power_domain` command for the corresponding power domain(s), then that condition will be used.

If this option is specified, the rule must be implemented whether or not the `-required` option is specified.

`-save_level` *`expr`*

Specifies that the states are saved when the save expression is `true` and the power is on.

The expression (*`expr`*) can be a Boolean function of the pin or port that directly controls the save operation.

If you specify this option with the `create_state_retention_rule` and the `-default_save_level` option with the `create_power_domain` command, the option specified with this command takes precedence.

If you omitted both the `-restore_level` option and the `-save_level` option, but you specified the `-default_restore_level` and `-default_save_level` options with the `create_power_domain` command for the corresponding power domain(s), then these conditions will be used.

If this option is specified, the rule must be implemented whether or not the `-required` option is specified.

`-save_precondition` *`expr`*

Specifies an additional condition that must be met for the save operation to be successful. The condition can be related to the clock, set and reset signals, but should not include signals that directly control the save and restore function of the cell.

If this option is not specified, the save operation is performed when the `save_edge` (`save_level`) condition evaluates to `true`.

The operands of the expression (*`expr`*) can be pins, ports or nets. The pins or ports that control the save or restore operation should not be included in this expression.

This option is only intended for RTL simulation.

`-secondary_domain` *`domain`*

Specifies the name of the power domain that provides the continuous power when the retention logic is in retention mode.

During implementation, the primary power and ground nets of this domain must be connected to the non-switchable power and ground pins of the state retention cells.

`-target_type {flop | latch | both}`

Specifies the type of sequential elements to convert to state retention registers.

If you specify `both`, then sequential elements of type `flop` and `latch` can be converted to state retention registers.

Default: `flop`

`-use_secondary_for_output`

Specifies that the output of the retention logic must retain the previous value when the primary domain is shut off and the secondary domain is on. In this case, the output pin of the instance is associated with the secondary domain of the instance.

By default, the output of the retention logic is corrupted when the primary domain is shut off and the secondary domain is on. In other words, the output of the instance is associated with the primary domain of the instance.

If both the primary and the secondary domain are off, then the output will be corrupted.

Example

In the following example, module `IPBlock` is instantiated as instance `IPInst` in design `Top`. Instance `IPInst` is part of a switchable power domain `X`.

`IPblock.cpf` is the CPF file for `IPBlock`:

```
set_design IPBlock
create_state_retention_rule -name sr1 -instances *
end_design
```

Consider the following 2 CPF files for the design Top:

1. Case 1: Top.cpf

   ```
   set_design Top
   create_power_domain -name X -instance IPInst -shutoff_condition switch_en \
   -default_restore_edge restore_en
   set_instance IPInst
   include IPBlock.cpf
   ```

   In this case, all instances of `IPInst` will be converted to state-retention flops using the restore condition specified for domain X at the top-level design.

2. Case 2: Top.cpf

```
set_design Top
create_power_domain -name X -instance IPInst -shutoff_condition switch_en
set_instance IPInst
include IPBlock.cpf
```

The state retention rule created for `IPBlock` is ignored because the `-restore_edge` option is not defined with the `create_state_retention_rule` command, nor is the `-default_restore_edge` option defined with the `create_power_domain` command for domain X.

Related Information

State Retention Cell on page 20

How to Model Different Types of State Retention Control on page 107

create_power_domain on page 168

define_state_retention_cell on page 313

update_state_retention_rules on page 279

define_library_set

```
define_library_set
     -name library_set
     -libraries list
     [-user_attributes string_list]
```

Creates a library set.

Options and Arguments

`-libraries` *`list`*

Specifies a list of library files (.lib files) or library groups.

A library group is a list of library files enclosed in braces.

Use a library group if you want to interpolate voltage values from libraries characterized for different voltages.

Note: File lists cannot contain wildcards.

`-name` *`library_set`*

Specifies the name of a library set.

Note: The specified string cannot contain wildcards.

`-user_attributes` *`string_list`*

Attaches a list of user-defined attributes to the library set. Specify a list of strings.

Example

```
define_library_set -name PD1_set -libraries {pads_1v.lib {ss_1v.lib ss_1v2.lib} }
```

Related Information

Library Group on page 16

Library Set on page 16

end_design

```
end_design
     [power_design_name]
```

Used with a `set_design` or `update_design` command, groups a number of CPF commands defining power intent that will be applied to the top module or to instances of one or more logic modules.

Options and Arguments

power_design_name	Specifies the name of the power design used with either the `set_design` or `update_design` command. **Note:** The name must match the name specified in the immediately preceding `set_design` or `update_design` command.

Example

```
Top
    Inst_A (mod_A)
        Inst_B (mod_B)
```

The names in parentheses are the corresponding module names.

```
set_design mod_B
# the following commands apply to hierarchy under mod_B
...
end_design mod_B
set_design Top
# the following commands apply to Top
...
# change scope to Inst_A
set_instance Inst_A
set_design mod_A
# the following command apply to hierarchy under mod_A
...
set_instance Inst_B -design mod_B
# the following commands still apply to hierarchy under mod_A
...
# change scope to Top
end_design mod_A
# the following commands apply to Top
...
```

```
end_design Top
```

Related Information

CPF Modeling for Hierarchical Design on page 74

set_design on page 212

set_instance on page 226

update_design on page 254

end_macro_model

```
end_macro_model
     [macro_mode_name]
```

Used with a `set_macro_model` command, groups a number of CPF commands that define power intent that will be applied to instances of one or more macro cells.

Options and Arguments

macro_model_name	Specifies the name of the macro model used with the `set_macro_model` command. **Note:** The macro model name must match the name specified in the immediately preceding `set_macro_model` command.

Related Information

Modeling a Macro Cell on page 65

set_macro_model on page 232

end_power_mode_control_group

```
end_power_mode_control_group
```

Used with a `set_power_mode_control_group` command, groups a set of CPF commands that define the power modes and power mode transitions that apply to the group defined by the preceding `set_power_mode_control_group` command.

Related Information

Power Mode Control Groups on page 81

set_power_mode_control_group on page 237

find_design_objects

```
find_design_objects pattern
     [-pattern_type {name | cell | module}]
     [-scope hierchical_instance_list]
     [-object {inst | port | pin | net}] [-direction {in | out | inout}]
     [-leaf_only | -non_leaf_only ]
     [-hierarchical] [-exact | -regexp] [-ignore_case]
```

Searches for and returns design objects that match the specified search criteria in the specified scope. This provides a general way to select design objects for use in CPF commands. All objects are returned with respect to the current scope.

Options and Arguments

`-direction {in | out | inout}`

Returns only pins or ports with the specified direction.

Applies only if the `-object` option is specified with `pin` or `port`. This option is ignored when used with other object types, and tools can choose whether or not to report a warning.

When you omit this option and you specify the `-object` option with either `pin` or `port`, all matching pins or ports will be returned, irrespective of the direction.

`-exact`

Returns only objects whose names exactly match the pattern value.

No wildcard expansion is performed.

`-hierarchical`

Specifies to perform a recursive search in the specified scopes and all children of those scopes.

By default, the search only applies to the specified scopes, not including its children.

`-ignore_case`

Specifies to perform a case insensitive search for both the *`pattern`* and the string specified by the `-scope` option.

By default, the search pattern is case sensitive.

`-leaf_only | -non_leaf_only`

Specifies that only instances without children `(-leaf_only)`, or with children (`-non_leaf_only`) should be returned.

Applies only if the `-object` option is specified with `inst` or if the `-object` option is omitted. Either of these options is ignored when used with an object type other than `inst`, and tools can choose whether or not to report a warning.

By default, all matched inst objects are returned.

`-object {inst | port | pin | net}`

Specifies the type of design objects to be returned.

- `inst`—Returns instances.
- `port`—Returns ports of the modules in the current scope.

  When you specify this type, the `-scope` and `-hierarchical` options are ignored. Some tools might report a warning in this case.
- `pin`—Returns pins.
- `net`—Returns nets.

Default: `inst`

`pattern` Specifies the search string. By default, it is a Tcl global expression.

`-pattern_type {name | cell | module}`

Specifies the type of name to be matched by the pattern string.

- `name`—Returns objects whose names match the pattern.

  This pattern type is allowed for any object type.
- `cell`—Returns objects that are instances of library cells whose names match the pattern.

  This pattern type is only allowed if `-object` is specified with the `inst` value or if the `-object` option is omitted.
- `module`—Returns objects that are instances of modules whose names match the pattern.

  This pattern type is only allowed if `-object` is specified with the `inst` value or if the `-object` option is omitted.

Default: `name`

`-regexp` Indicates that the specified *`pattern`* is a regular expression.

`-scope` *`hierarchical_instance_list`*

Specifies the scope or scopes from which the search must start. You can only reference the current scope and its children.

You can specify a list of hierarchical instances, but you cannot use wildcard characters or regular expressions.

When omitted, the search begins in the current scope.

It is an error if the specified hierarchical instances cannot be found within the current scope.

Examples

Assume the current scope is `top` with the following hierarchy:

```
top
    a
        a (inst of cell AND)
```

```
find_design_objects a
```

Returns `a`

```
find_design_objects a -hierarchical
```

Returns `a` and `a.a`

```
find_design_objects a -hierarchical -leaf_only
```

Returns `a.a`

```
find_design_objects a -hierarchical -non_leaf_only
```

Returns `a`

```
find_design_objects a -object inst -hierarchical R
```

Returns `a` and `a.a`

```
find_design_objects AND -hierarchical -object inst -pattern_type cell
```

Returns `a.a`

```
find_design_objects AND -hierarchical -object inst -pattern_type module
```

Returns `{}`

Related Information

Instance on page 13

Module on page 13

Net on page 13

Pin on page 14

Port on page 14

scope

get_parameter

```
get_parameter parameter_name
```

Returns the value of a predefined parameter in the current design.

An error message will be given if the parameter is not defined for the current design.

Options and Arguments

parameter_name	Specifies a parameter name. The parameter must have been defined with the `-parameters` option of the `set_design` command for the current design.

Example

IP block `fooMod` has two parameter values defined, parameter `x` with default value `0` and parameter `y` with default value `on`. When the module is instantiated as `fooInst`, the corresponding CPF model is also loaded but the parameter value of `x` is changed to `1` and the parameter value of `y` is changed to `off`.

```
set_design fooMod -ports ... -parameters { {x 0} {y on} }
....
if { [get_parameter y] == "off" } {
    if { [get_parameter x] == 0} {
        create_isolation_rule ... -output_type 0
    } else {
        create_isolation_rule ... -output_type 1
    }
}
end_design
set_design top
...
set_instance fooInst -design fooMod -domain_mapping ... \
-port_mapping ... -parameter_mapping { { x 1} { y off}}
...
end_design
```

Related Information

set_design on page 212

set_instance on page 226

identify_always_on_driver

```
identify_always_on_driver
     -pins pin_list [-no_propagation]
```

Specifies a list of pins in the design that must be driven by always-on buffer or inverter instances.

By default, all the inverter or buffer drivers of the transitive fanin cone of the specified pins are considered as always-on driver as well.

Options and Arguments

`-no_propagation`

Considers only the leaf-level driver of the specified pin must be the output pin of an always-on buffer or inverter instance.

`-pins` *`pin_list`*

Specifies the names of top-level ports or instance pins.

In case of a pin, specify the full hierarchical path of the pin.

Note: A bidirectional pin can also be considered as a driving pin.

identify_power_logic

```
identify_power_logic
     -type isolation
     {-instances instance_list | -module name}
```

Identifies any generic logic used for isolation that is instantiated in RTL or the gate-level netlist.

You can identify the isolation logic by listing all instances, or if all instances are instantiations of the same module, you can specify the module name.

Note: Any instances of special low power cells (such as level shifter cells, isolation cells, and so on) instantiated in RTL or the gate-level netlist that are defined through any of the library cell-related CPF commands are automatically identified.

Options and Arguments

`-instances` *instance_list*
: Specifies the names of all instances of the power logic selected through the `-type` option.

`-module` *name*
: Specifies the name of the RTL module whose instances represent the isolation logic.

`-type`
: Specifies the type of power logic to be identified.

  Currently, the only valid option is `isolation`.

Example

Assume the following Verilog module is used in the design for the purpose of isolation:

```
module iso (in, out, en);
input [7:0] in;
output [7:0] out;
    assign out = en ? 1 : in;
endmodule
```

To identify the isolation logic in the design, use the following command in the CPF file:

```
identify_power_logic -type isolation -module iso
```

identify_secondary_domain

```
identify_secondary_domain
     -secondary_domain domain
     {-instances instance_list | -cells cell_list }
     [ -domain power_domain [-from power_domain | -to power_domain]]
```

Identifies the secondary power domain for the selected instances with multiple power and ground pins. The primary power and ground nets of this secondary power domain are connected to the non-switchable power and ground pins of the identified instances.

By default, the secondary power domain for any instance that has both switchable and non-switchable power and ground pins is the secondary power domain defined for the power domain that the instance belongs to.

If several occurrences of this command apply to the same design object, the `-instances` option has a higher priority than the `-cells` option, which in turn has a higher priority than the `-domain` option.

Note: The targeted instances are instances that already existed in the netlist, that is, before the tools insert new low power logic based on the CPF rules.

Options and Arguments

`-cells` *`cell_list`*

Identifies the secondary power domain for the instances of the specified library cells.

The cells must be defined with both switchable power and ground pins and non-switchable power and ground pins. Examples of these types of cells are always on cells, power switch cells, retention cells, and so on.

`-domain` *`power_domain`*

Limits the selected instances to the instances that are part of the specified power domain.

The power domain must have been previously defined with the `create_power_domain` command.

`-from` *power_domain*

Limits the selected instances to the instances that are

1. Part of the domain specified with the `-domain` option
2. Driven by instances in the domain specified by this option.

The power domain must have been previously defined with the `create_power_domain` command.

`-instances` *instance_list*

Specifies the list of the selected instances. If the selected instance is a hierarchical instance, all leaf instances and registers in its hierarchy but not of its children are selected.

`-secondary_domain` *domain*

Specifies the name of the secondary power domain. The power domain must have been previously defined.

Note: The specified string cannot contain wildcards.

`-to` *power_domain*

Limits the selected instances to the instances that are

1. Part of the domain specified with the `-domain` option
2. Driving instances in the domain specified by this option.

The power domain must have been previously defined with the `create_power_domain` command.

Related Information

Secondary Power Domain on page 18

include

```
include file
```

Includes a CPF file or a Tcl file within a CPF file.

The path name to the file can contain a period (.) to refer to current directory of the CPF file being read and ".." to refer to the parent directory of the current directory of the CPF file being read.

Note: This command differs from the `source` command in Tcl syntax where a period (.) always refers to the working directory of the tool reading the top level CPF file and ".." refers to the parent of the working directory of the tool reading the top level CPF file.

Options and Arguments

file	Specifies the path name of the file to be included.

Examples

Consider the following file structure:

```
home
    | models
        | impl  --> impl.cpf
        | sim  --> sim.cpf
    | tmp  --> top.cpf
        | IP_block  --> IP.cpf
        | tech  --> tech.cpf
```

- To include the contents of the `tech.cpf` and `IP.cpf` files in the top level CPF file (top.cpf), use the following two commands:

  ```
  include ./tech/tech.cpf
  include ./IP_block/IP.cpf
  ```

  The path names of these two CPF files are defined with respect to the location (current directory) of the `top.cpf` file.

- To include the contents of the `sim.cpf` and `impl.cpf` files in the `IP.cpf` file, use the following two commands:

  ```
  include ../../models/sim/sim.cpf
  include ../../models/impl/impl.cpf
  ```

  The path names of these two CPF files are defined with respect to the location of the `IP.cpf` file.

set_analog_ports

```
set_analog_ports port_list
     [-user_attributes string_list]
```

Identifies a list of top-level analog ports in a power design or a list of analog ports in a macro model.

The specified analog ports must be connected to other ports or pins that were declared as analog ports by the same command or as analog pins in a pad cell.

Analog ports can be associated with a power domain of a macro.

Options and Arguments

port_list

Specifies a list of analog signal ports.

Note: Do not include ports that are driving or are driven by analog circuitry if they can be connected to a digital port.

Note: Do not declare power and ground pins of a macro model as analog ports, because they are treated as analog ports by default.

-user_attributes *string_list*

Associates a string or list of strings with an analog port.

These strings can be used to check if the ports are connected to ports with the same strings.

Related Information

define_pad_cell on page 303

set_macro_model on page 232

end_macro_model on page 197

set_array_naming_style

```
set_array_naming_style
     [string]
```

Specifies the format used to name the design objects in the netlist starting from multi-bit arrays in the RTL description. For sequential elements, the bit information is appended to the instance name which is determined by the `set_register_naming_style` command.

The command returns the new setting or the current setting in case the command was specified without an argument.

Options and Arguments

string	Specifies the format of the bit information. The string must have the following format: [*character*]%d[*character*] You can use angle brackets, square brackets, or underscores. *Default:* `\[%d\]`

Example

```
set_array_naming_style <%d>
```

Related Information

Specifying the Representation of the Bits on page 29

Information Inheritance on page 120

set_cpf_version

```
set_cpf_version
     [value]
```

Specifies the version of the format.

The command returns the new setting or the current setting in case the command was specified without an argument.

If specified, this command must be the first CPF command in a CPF file.

Important

You cannot have multiple occurrences of this command with different values set in the same scope.

Multiple occurrences are allowed in different scopes, but the CPF version of a lower scope cannot be newer than the CPF version of the parent scope.

if your design has different versions of CPF files, the semantics of the newest version takes precedence.

Options and Arguments

value	Specifies the version. Use a string. Currently allowed values are `1.0`, `1.0e`, `1.1`, and `2.0`. *Default*: `1.1`

Example

```
set_cpf_version 1.1
```

set_design

```
set_design
     power_design [-modules module_list]
     [-ports port_list] [-input_ports port_list]
     [-output_ports port_list] [-inout_ports port_list]
     [-honor_boundary_port_domain]
     [-parameters parameter_value_list]
     [-testbench] ...
```

Begins the definition of a power design, identified by a name that is unique among power designs within the same scope.

The power intent represented by the power design is defined by the CPF commands between this command and a matching `end_design` command.

A power design can be applied to a top module or to instances of one or more logic modules.

Options and Arguments

`-honor_boundary_port_domain`

Specifies to treat each boundary port domain assignment as a design constraint at the top level after the block is instantiated at the top. At the top level, each corresponding hierarchical pin becomes a virtual leaf level driver or leaf level load and its power domain corresponds to the power domain definition for the boundary port. In this case, the netlist traversal stops at this pin.

If this option is omitted, the tools should traverse through the hierarchical pin to find the real leaf level driver or leaf level load of the net connected to the hierarchical pin.

Note: This option only applies to the primary inputs and primary outputs of the current design.

`-inout_ports` *`port_list`*

Specifies a list of *virtual* inout ports in a module to which this power design applies.

`-input_ports` *`port_list`*

Specifies a list of *virtual* input ports in a module to which this power design applies.

`-modules` *`module_list`*

Specifies the names of the modules to which the power design applies. The power design can only be applied to instances of a logic module whose name exactly matches one of the specified names.

When omitted, the power design can be applied to instances of any logic module.

The module names cannot contain wildcards.

`-output_ports` *`port_list`*

Specifies a list of *virtual* output ports in a module to which this power design applies.

`-parameters` *`parameter_value_list`*

Specifies a list of parameters with their default values.

Use the following format for each parameter:

{*`parameter_name default_value`*}

The default value can be a number of a string.

`-ports` *`port_list`* Specifies a list of *virtual* ports in the specified module.

Virtual ports do not exist in the definition of this module but will be needed for the control signals of the low power logic such as isolation logic, state-retention logic, and so on.

Note: For backward compatibility, `-ports` is the same as `-input_ports`. `-ports` will be obsolete in a future release.

`power_design` Specifies a name for the power intent described by this command.

The name must be unique among power designs created in the same scope.

`-testbench` Specifies that the power design applies to a testbench module.

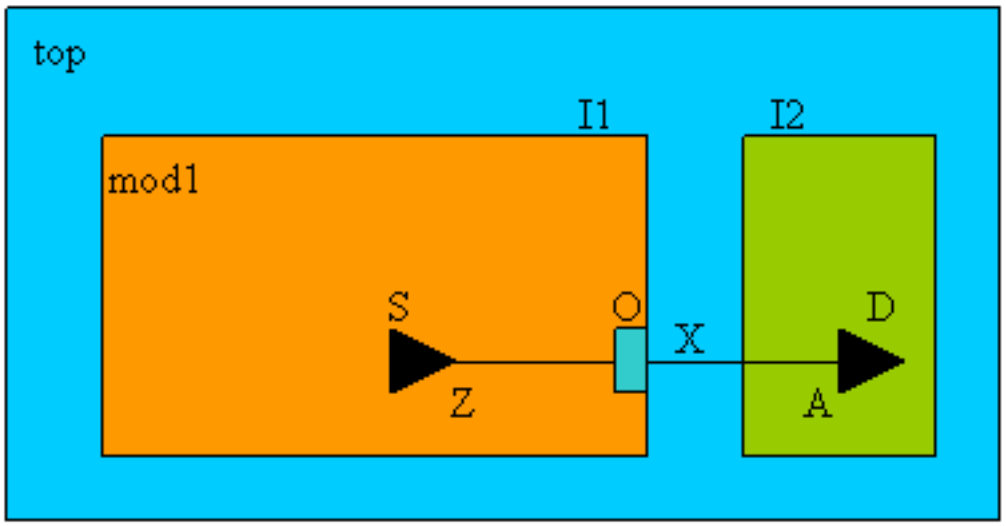
Use this option to create a testbench-level CPF to drive power domains for designs under test, without the need to create power domains or modes in the testbench level CPF.

Note: You can only specify a top-level design with this option. That is, the model cannot be a hierarchical CPF model, loaded after a `set_instance` command.

Note: When specified, the ports of the design under test (DUT) module must be treated as the leaf level drivers or loads when considering the isolation or level shifter rules in the CPF for the DUT module.

Examples

- In the following example, module `mod1` is an IP block and has a default switchable domain. The output port `O` is declared as part of a virtual domain that is always on.




```
set_design mod1 -honor_boundary_port_domain
create_power_domain -name PDVirtual -boundary_ports O
create_power_domain -name PDCore -default -shutoff_condition en \
  -base_domains PDVirtual
end_design mod1
set_design top
create_power_domain -name PDAO -default
create_power_domain -name PD2 -shutoff_condition en2
set_instance I1 -design mod1 -domain_mapping {{PDVirtual PDAO}}
end_design
```

When module `mod1` is instantiated at the top, the block-level domain `PDVirtual` is mapped into the top level domain `PDAO`. Because block `mod1` is specified with the `-honor_boundary_port_domain` option, the boundary domain assignment of port `O` to domain `PDAO` (due to the domain mapping) should be treated as a design constraint at the top level, even though the hierarchical pin I1/O has a leaf level driver `I1/S/Z` in domain `I1/PDCore` and a leaf level load `I2/D/A` in domain `PD2`). The implementation tool can insert a buffer for the top net `x` within power domain `PDAO` to meet the design requirement.

- IP block `fooMod` has two parameter values defined, parameter `x` with default value `0` and parameter `y` with default value `on`. When the module is instantiated as `fooInst`, the corresponding CPF model is also loaded but the parameter value of `x` is changed to `1` and the parameter value of is `y` changed to `off`.

```
set_design fooMod -ports ... -parameters { {x 0} {y on} }
....
if { [get_parameter y] == "off" } {
    if { [get_parameter x] == 0} {
        create_isolation_rule ... -output_type 0
    } else {
        create_isolation_rule ... -output_type 1
    }
}
end_design

set_design top
...
set_instance fooInst -design fooMod -domain_mapping ... \
-port_mapping ... -parameter_mapping { { x 1} { y off}}
...
end_design
```

- Instance `I1` is instantiated from logic module `foo`, but after synthesis and module uniquification, the logic module is renamed to `foo_1`. Using the `-modules` option in combination with the `find_design_objects` command, the following hierarchical CPF is legal:

```
set_design my_power_design -modules { [find_design_objects -type module foo* ]}
....
end_design my_power_design
set_instance I1 -design my_power_design ...
```

Related Information

Power Design on page 17

Virtual Port on page 18 in

end_design on page 195

find_design_objects on page 199

get_parameter on page 203

set_instance on page 226

update_design on page 254

set_diode_ports

```
set_diode_ports
     {-positive pos_port_list -negative neg_port_list
     |-positive pos_port_list | -negative neg_port_list}
```

Specifies a list of ports of a macro cell that connect to the positive and negative pins of a diode inside a macro cell.

When you specify both the `-positive` and `-negative` options, the specified macro model ports are connected through diodes. You can specify a single positive port and multiple negative ports, or multiple positive ports and one negative port. You cannot specify multiple positive and multiple negative ports when both options are used.

When you specify only the `-positive` option, each port specified is connected to the anode of a power clamp diode and the cathode of the diode is connected to the primary power of the associated domain of the port.

When you specify only the `-negative` option, each port specified is connected to the cathode of a ground clamp diode and the anode of the diode is connected to the primary ground of the associated domain of the port.

A port can appear in multiple `set_diode_ports` commands. However, it is an error if the same port appears in both the `-negative` and `-positive` options of the same command.

Note: A port can be declared as a member of a power domain and as a diode port. If a port is only declared as a diode port and not associated with a power domain, it is considered to be a floating port.

Options and Arguments

`-negative` *`port_list`*

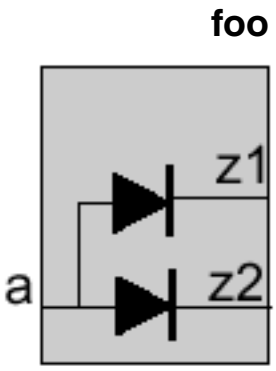
Specifies the names of the ports that connect to the negative inputs or cathode of a diode.

`-positive` *`port_list`*

Specifies the names of the ports that connect to the positive inputs or anode of a diode.

Examples

- The following macro cell has two diodes.




```
set_macro_cell foo
...
set_diode_ports -positive a -negative {z1 z2}
...
end_macro_cell
```

- The following commands connect input port `A` to a power and ground clamp diode:

```
set_diode_ports -negative A
set_diode_ports -positive A
```

- The following command causes an error because input port A is specified in the `-positive` and `-negative` options of the same command.

```
set_diode_ports -positive {A B} -negative {A}
```

Related Information

set_macro_model on page 232

set_equivalent_control_pins

```
set_equivalent_control_pins
     -master pin
     -pins pin_expression_list
     { -domain domain| -rules rule_list }
```

Specifies a list of pins that are equivalent with a master control pin. The master control pin is part of the definition of a shutoff condition, isolation condition or state retention condition. The referred condition can contain only the master control pin in its expression.

The following requirements apply to the pins:

- A master control pin cannot appear as an equivalent control pin in another `set_equivalent_control_pins` command.
- An equivalent control pin can only be associated with one master control pin.

A common usage of equivalent control pins is to generate time-staged control signals (control signals with different arrival times) to control the rush current when simultaneously turning on multiple devices at once.

Options and Arguments

`-domain` *power_domain*

Specifies the name of the domain for which the master control pin is part of either

- the shutoff condition
- the active state condition
- the default isolation condition
- the default save/restore condition
- the enable condition for a power switch rule (for verification only)

The power domains must have been previously defined with the `create_power_domain` command.

`-master` *pin*

Specifies the name of the master control pin.

You can specify a port or an instance pin.

`-pins` *`pin_expression_list`*

Specifies the pins that are equivalent to the master control pin and that drive staged control signals with respect to the control signal driven by the master control pin.

You can specify ports and instance pins and the negation operator to indicate the inverted polarity of the pin or port.

The list can contain wildcards, but the wildcard cannot be used with the negation (!) operator.

`-rules` *`rule_list`* Specifies the rules for which you specify the equivalent control pins. You can only list isolation rules and state retention rules that are considered complete.

The list can contain wildcards.

Examples

- In the following example, the actual shutoff condition for domain X for simulation is `!PDN1 | !PDN2 | PDN3`.

```
create_power_domain -name X -shutoff_condition !PDN1  \
  -active_state_condition { 1.0v@en1 1.1v@!en1 }
set_equivalent_control_pins -master PDN1 -pins {PDN2 !PDN3}   -domain X
set_equivalent_control_pins -master en1 -pins {!en2 en3}   -domain X
```

  The condition for domain X to be at nominal condition 1.0v is `en1 &!en2 & en3.`

  The condition for domain X to be at nominal condition 1.1v is `!en1 & en2 &!en3.`

- In the following example, assume only an AND type isolation cell is available.

```
create_isolation_rule -name myISO -from PD1 -to PD2 -isolation_output low \
-isolation_condition pcm/A
```

  Without the presence of equivalent pins, the synthesis tool needs to insert an inverter on the isolation enable line because the isolation logic is enabled when the `pcm/A` signal is high, while the required output of the isolation gate is `low`.

  When an equivalent pin with opposite polarity is present, the synthesis tool can use this equivalent control pin and as a result no inverter is required on the control line. In the example below, the synthesis tool can use `pcm/B` whose polarity is the negation of `pcm/A` as the control driver.

```
set_equivalent_control_pins -master pcm/A -pins {!pcm/B pcm/C} -rules myISO
```

Related Information

Chapter 6, “Precedence and Semantics of the Rules”

create_isolation_rule on page 145

create_power_domain on page 168

create_state_retention_rule on page 186

set_floating_ports

```
set_floating_ports port_list
```

Specifies a list of ports of a macro cell that are not connected to any logic inside the macro cell.

These ports are excluded from any power domain defined for the macro cell.

Options and Arguments

port_list	Specifies the names of the ports that can be floating. The ports must be valid ports of the macro cell. The list can contain wildcards.

Related Information

end_macro_model on page 197

set_input_voltage_tolerance on page 223

set_macro_model on page 232

set_wire_feedthrough_ports on page 253

set_hierarchy_separator

```
set_hierarchy_separator
     [character]
```

Specifies the hierarchy delimiter character used in the CPF file.

The command returns the new setting or the current setting in case the command was specified without an argument.

Options and Arguments

character	Specifies the hierarchy delimiter character. *Default:* .

Related Information

Hierarchy Delimiter on page 26

Information Inheritance on page 120

set_input_voltage_tolerance

```
set_input_voltage_tolerance
     { -power lower[:upper] | -ground [lower:]upper
     | -power lower[:upper] -ground [lower:]upper }
     [-domain power_domain] [-pins pin_list]
```

Specifies the input supply voltage tolerance constraint at the selected input pins. If the driver and receiver voltage difference is more than the tolerance voltage, a power (ground) level shifter is required.

Note: If the command is used In the hierarchical flow without `-pins` option, the specified voltage tolerance only applies to the selected pins of the scope in which the command is used.

Options and Arguments

`-domain` *power_domain*

Indicates that the specified voltage tolerance applies to the input pins of the specified power domain.

The power domains must have been previously defined with the `create_power_domain` command.

Note: If this option is specified with the `-pins` option, the pins that are not input pins of this domain will be ignored.

`-ground [`*lower*`:]`*upper*

Specifies a lower and upper bound for the ground voltage tolerance of the selected pins.

The following relation applies:

$$voltage_{receiver} + lower \leq voltage_{driver} \leq voltage_{receiver} + upper$$

where

$voltage_{receiver}$ is the ground voltage of the receiving power domain

$voltage_{driver}$ is the ground voltage of the driving power domain

If the relationship is violated a ground up-shifter or ground down-shifter is required.

- *lower* must be smaller than or equal to zero.

  Default: negative infinity

- *upper* must be larger than or equal to zero.

  Default: 0

Note: The default values apply to all input pins of the current design.

`-pins` *pin_list*

Specifies the input pins or top-level output ports to which the voltage tolerance applies.

Note: If you specify neither this option, nor the `-domain` option, the voltage tolerance applies to all input pins in the current scope.

`-power` *lower*`[:`*upper*`]`

Specifies a lower and upper bound for the power voltage tolerance of the selected pins.

The following relation applies:

$$voltage_{receiver} + lower \leq voltage_{driver} \leq voltage_{receiver} + upper$$

where

$voltage_{receiver}$ is the power voltage of the receiving power domain

$voltage_{driver}$ is the power voltage of the driving power domain

If the relationship is violated a power up-shifter or power down-shifter is required.

- *lower* must be smaller than or equal to zero.

  Default: 0

- *upper* must be larger than or equal to zero.

  Default: positive infinity

Note: The default values apply to all input pins of the current scope.

Examples

- The following command specifies that the driving power voltage can be 0.1 less or 0.3 larger than the voltage at the input pins without the need for a level shifter.

  ```
  set_input_voltage_tolerance -power -0.1:0.3
  ```

- The following command specifies that the driving power voltage can be 0.1 less than the voltage at the input pins without the need for a level shifter, but does not give a requirement for a high to low shifter.

  ```
  set_input_voltage_tolerance -power -0.1
  ```

- The following command specifies that the driving power voltage can be 0.3 volt higher than the receiver voltage without a level shifter, but there is always a requirement for low to high shifting.

  ```
  set_input_voltage_tolerance -power 0:0.3
  ```

- The following command specifies that the driving ground voltage can be 0.3 less or 0.1 larger than the receiving ground voltage without requiring a ground level shifter.

  ```
  set_input_voltage_tolerance -ground -0.3:0.1
  ```

- The following command specifies that the driving power voltage can be 0.2 less or 0.4 larger than the receiving voltage without requiring a power level shifter, and that the driving ground voltage can be 0.3 less or 0.1 larger than the receiving ground voltage without requiring a ground level shifter.

  ```
  set_input_voltage_tolerance -power -0.2:0.4 -ground -0.3:0.1
  ```

Related Information

define_level_shifter_cell on page 293

end_macro_model on page 197

set_macro_model on page 232

set_instance

```
set_instance
     [ instance
       [-design power_design | -model macro_model]
       [-port_mapping port_mapping_list]
       [-domain_mapping domain_mapping_list]
       [-parameter_mapping parameter_mapping_list] ]
```

Changes the scope to the specified instance or links a previously defined CPF power design or macro model to the specified instance.

The command returns the current scope in case the command was specified without an argument. If the current scope is the top design, the hierarchy separator is returned.

If the command is specified with an instance name only, it will change the design scope for the purpose of searching design objects referenced in the commands after it.

If the command is specified with an instance name, and without the `-design` or `-model` options, but with some other options, it must be followed by a `set_design` command or a `set_macro_model` command. Some commands are allowed between `set_instance` and `set_design` or `set_macro_model`. These commands are shown in the Command Dependency table before the `set_design` command.

The scope is used for naming resolution and affects

- All design objects
- All the expressions in the CPF design-related constraints

All CPF objects referred to in the library cell-related CPF commands are scope *insensitive*.

Any rule created in the block-level CPF file that references a virtual port that is declared using the `-inout_ports`, `-input_ports`, `-output_ports` or `-ports` option of the `set_design` command, but whose mapping is not specified through the `-port_mapping` option, will be ignored.

Options and Arguments

`-design` *power_design*

Specifies to link a previously defined power design to the specified instance. The scope is not changed when this option is used.

A bound power design is associated with either

- an instance through the set_instance command
- the top level module of a design

To bind a power design to the top level module of a design,

- you can specify the power design name at the command level when reading or elaborating the CPF
- a tool can select the only power design that is not bound to any instance.

It is an error to bind multiple power designs to the same instance.

`-domain_mapping` *domain_mapping_list*

Specifies the mapping of the domains in the current scope to the domains for the specified design or macro model.

Use the following format to specify a domain mapping:

{*domain_in_child_scope domain_in_parent_level_scope*}

The power domains must have been previously defined with the `create_power_domain` command.

instance

Specifies an instance. The instance must be a valid instance in the current scope.

If the name is not followed by either the `-design` or `-model` option, the `set_instance` command changes the scope to the specified instance.

`-model` *`macro_model`*

Specifies to use a previously loaded CPF description for the specified macro model.

This description must precede the current `set_instance` command and is contained between a `set_macro_model` *`macro_model`* command and the next `end_macro_model` command.

In this case, the scope does not change.

`-parameter_mapping` *`parameter_mapping_list`*

Specifies the mapping of the parameters specified in the `set_design` command to the local values.

Note: It is an error if the parameter is not specified in the `set_design` command.

Use the following format to specify a parameter mapping:

{*`parameter_name local_value`*}

`-port_mapping` *`port_mapping_list`*

Defines the connection between a pin of the specified instance and a pin or port visible in the current scope.

The instance pin must be a pin corresponding to the virtual port defined in the `set_design` command or a real pin from the cell or module definition.

Note: It is an error if the specified connection conflicts with any existing netlist connection.

Use the following format to specify a port mapping:

{*`block_instance_pin current_scope_driving_pin`*}

{*`macro_port parent_level_driver`*}

Examples

- The following examples assume that `inst1_foo` and `inst2_foo` are instances of the same logic module `foo` under top level module `top`.
  - If both instances use the same power design:

```
set_design foo
...
end_design foo
set_design top
set_instance inst1_foo -design foo
set_instance inst2_foo -design foo
end_design top
```

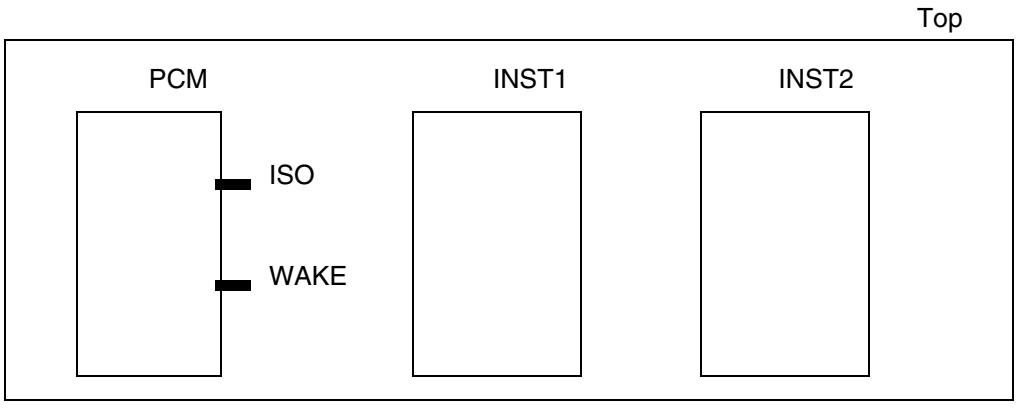
- If each instances uses its own power design:

```
set_design foo1
...
end_design foo1
set_design foo2
...
end_design foo2
set_design top
set_instance inst1_foo -design foo1
set_instance inst2_foo -design foo2
end_design top
```

- In the following example, `set_instance inst1_foo` changes the scope to `inst1_foo`. As a result,the power design `foo` is only defined for scope `inst1_foo`. As a result, it is an error to reference `foo` in the second `set_instance` command since the scope at this point is at the top level.

```
set_design top
...
set_instance inst1_foo -design foo
set_design foo
...
end_design foo
set_instance inst2_foo -design foo
end_design top
```

- The following example illustrates the use of the `-port_mapping` option.



Design `Top` has three hierarchical instances:

- `INST1`, core logic instantiated from module `mod1`, is part of a switchable power domain, and will use state retention logic
- `INST2`, core logic instantiated from module `mod1`, is part of a switchable power domain, but will not use state retention logic
- `PCM`, the power control block

  The `ISO` pin on the PCM block will control the isolation logic in instances `INST1` and `INST2`

  The `WAKE` pin on the PCM block will control the state retention logic in `INST1`.

Note: The RTL description of module `mod1` has no initial isolation logic or state retention logic.

CPF file for module `mod1` is called `mod1.cpf`, while CPF file for the design Top is called `Top.cpf`.

Within `mod1.cpf` rules for isolation logic and state retention logic are specified. Because the RTL description of module `mod1` does not contain ports for the isolation enable and the state retention restore signal, you need to declare the virtual ports `iso_en` and `restore`.

When this CPF file is applied to instance `INST1`, you must specify the port mapping for both virtual ports.

When this CPF file is applied to instance `INST2`, you must only specify the port mapping for the isolation enable port `iso_en`. In this case the state retention rule in the `mod1.cpf` file will be ignored.

```
#mod1.cpf
set_design mod1 -ports { iso_en restore )
...
create_isolation_rule ... -isolation_condition iso_en
create_state_retention_rule ... -restore_edge restore
...
end_design


#Top.cpf
set_design Top
...
set_instance INST1 -port_mapping {{iso_en PCM/ISO} {restore PCM/WAKE}}
include mod1.cpf
...

set_instance INST2 -port_mapping {{iso_en PCM/ISO}}
include mod1.cpf
...
end_design
```

- IP block `fooMod` has two parameter values defined, parameter `x` with default value `0` and parameter `y` with default value `on`. When the module is instantiated as `fooInst`, the corresponding CPF model is loaded but the parameter value of `x` is changed to `1` and the parameter value of is `y` changed to `off`.

```
set_design fooMod -ports ... -parameters { {x 0} {y on} }
....
if { [get_parameter y] == "off" } {
    if { [get_parameter x] == 0} {
        create_isolation_rule ... -output_type 0
    } else {
        create_isolation_rule ... -output_type 1
    }
}
end_design
set_design top
...
set_instance fooInst -design fooMod -domain_mapping ... \
-port_mapping ... -parameter_mapping { { x 1} { y off}}
...
end_design
```

Related Information

Design Objects on page 13

Power Design on page 17

CPF Modeling for Hierarchical Design on page 74

Command Dependency on page 116

set_design on page 212

set_macro_model

```
set_macro_model macro_model_name [-cells cell_list]
```

Indicates the start of the CPF content of a custom IP.

A simulation tool that has a behavioral simulation model can take the full description of the macro model and perform power-aware simulation. Tools for static verification and implementation can only use the interface definitions because to them, the macro model is a blackbox.

Only following CPF commands are allowed in a macro model definition:

```
create_isolation_rule
create_mode_transition
create_nominal_condition
create_power_domain
create_power_mode
create_power_switch_rule
create_state_retention_rule
set_analog_ports
set_diode_ports
set_floating_ports
set_input_voltage_tolerance
set_pad_ports
set_power_source_reference_pin
set_sim_control
set_wire_feedthrough_ports
update_power_domain
```

Commands issued from inside a macro model are not intended to drive the implementation of the macro, but rather they are intended to describe the behavior of the macro model.

Note: A macro model specification should explicitly associate each non power and ground port to a power domain, or declare it as a floating port, a feedthrough port, or a diode port.

Within a macro model definition, you cannot have another CPF model.

Options and Arguments

-cells cell_list

Lists the names of the cells to which the macro model applies. The model can only be applied to instances of a cell whose name exactly matches one of the specified names.

When this option is omitted, the macro model can be applied to instances of any library cell.

This option allows you to create one macro model representing a family of macro cells. This model can be reused for all instances of these cells. See the example using the -cells option, below.

The cell names cannot contain wildcards.

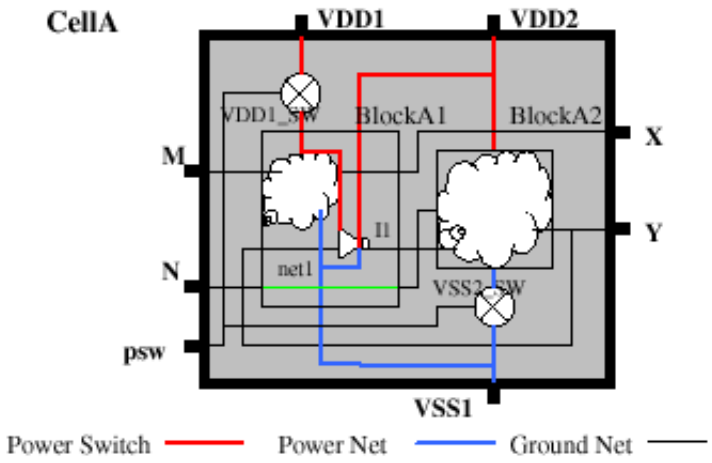
macro_model_name Specifies the name of the macro for which the CPF description follows.

Examples

- The following example illustrates the use of the `-cells` option to create a single macro model that applies to two different RAM cells.

```
set_macro_model generic_ram -cells { ram_single_port ram_dual_port }
...
end_macro_model
set_instance ram1 -model generic_ram ;# ram1 is instantiated from RAM cell
            # ram_single_port
set_instance ram2 -model generic_ram ;# ram2 is instantiated from RAM cell
            # ram_dual_port
```

- The following commands describe the power intent for macro cell CellA shown below.




```
set_macro_model CellA
create_power_domain -name PD_VDD1 -boundary_ports {iso[0] psw[0]} -default
create_power_domain -name PD_VDD2 -boundary_ports {iso[1] psw[1] srpg}
create_power_domain -name PD_VDD1_SW -boundary_ports {M X} \
-shutoff_condition { !psw[0] }
create_power_domain -name PD_VDD2_SW -boundary_ports {Y N} \
-shutoff_condition { !psw[1] }
create_power_switch_rule -name switch1 -domain PD_VDD1_SW \
-external_power_net VDD1
create_power_switch_rule -name switch2 -domain PD_VDD2_SW \
-external_power_net VDD2
update_power_domain -name PD_VDD1 -primary_power_net VDD1 \
-primary_ground_net VSS1
update_power_domain -name PD_VDD2 -primary_power_net VDD2 \
-primary_ground_net VSS1
end_macro_model
```

- Assume a RAM cell has 2 power pins and 1 ground pin. Power pin VDDA drives the memory array (mem in the behavioral model) and can be shut off externally. Power pin VDD drives the rest of the peripheral logic of the memory. When PDVDDA is on, PDVDD has to be on.

```
set_macro_model ram_256x8
create_power_domain -name PDVDD -default -boundary_ports { * }
create_power_domain -name PDVDDA -instances mem*
update_power_domain -name PDVDD -primary_power_net VDD -primary_ground_net VSS
update_power_domain -name PDVDDA -primary_power_net VDDA \
    -primary_ground_net VSS
create_nominal_condition -name on -voltage 1.0 -state on
create_power_mode -name on -domain_conditions { PDVDD@on PDVDDA@on }
create_power_mode -name sleep -domain_conditions { PDVDD@on PDVDDA@off }
end_macro_model ram_256x8
```

When the RAM cell is instantiated at the chip level, the PDVDDA domain can be mapped into a top-level switchable domain as illustrated below:

```
set_design top

create_power_domain -name PDtop1 -default
create_power_domain -name PDtop2 -instances ... -shutoff_condition ...

create_power_nets -nets "VDD1 VDD2"
create_ground_nets -nets "VSS"

update_power_domain -name PDtop1 -primary_power_net VDD1 \
    -primary_ground_net VSS
update_power_domain -name PDtop2 -primary_power_net VDD2 \
    -primary_ground_net VSS

set_instance -model RAMInst ram_256x8 \
    -domain_mapping { {PDVDD PDtop1} {PDVDDA PDtop2 } }

end_design top
```

According to the domain mapping, the memory array of the RAM cell now belongs to PDtop2, and the peripheral logic of the RAM cell and all of its boundary ports belong to PDtop1. For simulation, the memory array will be corrupted when PDtop2 is shut off. For implementation, tools can insert level shifter or isolation logic if there is a level shifter rule or isolation rule defined at the top level and there is domain crossing at the boundary pins of the RAM cell. Also, the power net VDD1 at the top level should be connected to the power pin VDD of the RAM cell, and the top level power net VDD2 should be connected to the power pin VDDA of the RAM cell.

Related Information

Modeling a Macro Cell on page 65

end_macro_model on page 197

set_floating_ports on page 221

set_input_voltage_tolerance on page 223

set_instance on page 226

set_wire_feedthrough_ports on page 253

set_pad_ports

```
set_pad_ports pin_list
```

Specifies a list of ports of a macro cell that connect directly to a bonding port at the chip level.

Use this command if the macro cell models a pad cell or the input of the macro cell has internal pad logic.

Options and Arguments

pin_list	Specifies the names of the ports that connect to bonding ports.

Related Information

create_analysis_view on page 132

set_macro_model on page 232

set_power_mode_control_group

```
set_power_mode_control_group -name group
     { -domains domain_list
     | -groups group_list
     | -domains domain_list -groups group_list}
```

Groups a list of power domains and other power mode control groups.

This command together with the `end_power_mode_control_group` command groups a set of CPF commands that define the power modes and power mode transitions that apply to this group only.

Only following CPF commands are allowed in a power mode control group definition:

```
create_analysis_view
create_mode_transition
create_power_mode
update_power_mode
```

Note: By default, all power modes and power mode transitions within a lower scope belong to an automatic defined power mode control group named after the scope.

Options and Arguments

`-domains` *`domain_list`*

Specifies the list of power domains controlled by the power control manager associated with the specified group.

The power domains must have been previously defined with the `create_power_domain` command.

`-groups` *`group_list`*

Specifies the list of power mode control groups whose power control manager is controlled by the power control manager associated with the specified group.

The specified group must have been previously defined with another `set_power_mode_control_group` command.

`-name` *`group`*

Specifies the name of the power mode control group.

Example

The following example defines a group `CORE` with three power domains.

```
set_power_mode_control_group  -name CORE  -domains {CPU FPU CACHE}
create_power_mode -name M1 -domain_conditions { CPU@low CACHE@on}
create_power_mode -name M2 -domain_conditions { CPU@high CACHE@on}
create_power_mode -name M3 -default -domain_conditions { CPU@high CACHE@on FPU@on}
create_mode_transition -name CT1 -from M1  -to M2 -start_condition ...
create_mode_transition -name CT2 -from M2  -to M3 -start_condition ...
create_mode_transition -name CT3 -from M3  -to M2 -start_condition ...
create_mode_transition -name CT4 -from M3  -to M1 -start_condition ...
end_power_mode_control_group
```

Related Information

Power Mode Control Groups on page 81

end_power_mode_control_group on page 198

set_power_source_reference_pin

```
set_power_source_reference_pin pin
     -domain power_domain -voltage_range min:max
```

Specifies an input pin of the macro cell as the voltage reference pin for a power source domain.

Options and Arguments

`-domain` *power_domain*

Specifies the name of the power source domain.

The domain must have been previously defined with the `-power_source` option of the `create_power_domain` command.

pin

Specifies the name of the voltage reference pin.

`-voltage_range` *min:max*

Specifies the required voltage range for the voltage reference pin. The range includes the minimum and maximum values.

Example

The following example declares pin `AVDD` as the voltage reference pin for power source domain `PDVOUT`.

```
set_macro_model regulator
create_power_domain -name PDVOUT -default -base_domains {PDVIN} -power_source
...
create_nominal_condition -name LDO_range -voltage 1.1 -pmos_bias_voltage {1.1 1.3}
...
create_power_mode -name PM -default -domain_conditions \
    { PDREF@REF PDVIN@HVDD PDVOUT@LDO_range }
..
set_power_source_reference_pin AVDD -domain PDVOUT voltage_range 1.0:1.1
...
end_macro_model regulator
```

Note: For proper working of the voltage regulator, the voltage values of pin `AVDD` must be between 1.0 and 1.1 Volt.

Related Information

Power Source Domain on page 18

Modeling a Voltage Regulator in the *Common Power Format User Guide*

set_macro_model on page 232

set_power_target

```
set_power_target
     { -leakage float | -dynamic float
     | -leakage float -dynamic float
```

Specifies the targets for the average leakage and dynamic power of the current design across all the power modes. All power targets must be specified in the units specified by the `set_power_unit` command.

Options and Arguments

`-dynamic` *float*	Specifies the target for the average dynamic power.
`-leakage` *float*	Specifies the target for the average leakage power.

Related Information

set_power_unit on page 242

set_power_unit

```
set_power_unit
     [pW|nW|uW|mW|W]
```

Specifies the unit for all power values in the CPF file.

The command returns the new setting or the current setting in case the command was specified without an argument.

Options and Arguments

`[pW|nW|uW|mW|W]`	Specifies the power unit. You can specify any of these five values. *Default*: `mW`

Related Information

Information Inheritance on page 120

set_power_target on page 241

set_register_naming_style

```
set_register_naming_style
     [string%s]
```

Specifies the format used to name flip-flops and latches in the netlist starting from the register names in the RTL description.

The command returns the new setting or the current setting in case the command was specified without an argument.

Options and Arguments

string	Specifies the suffix to be appended to the base name of a register. The %s represents the bit information. *Default*: _reg%s

Related Information

Specifying the Representation of the Base Name on page 28

Information Inheritance on page 120

set_sim_control

```
set_sim_control [-targets target_list [-exclude target_list]]
     {-action power_up_replay [-controlling_domain domain ]
     |-action disable_corruption -type { real | wreal | integer | reg | module |
        instance}
     |-action {disable_isolation | disable_retention} }
     [-domains domain_list | -instances instance_list]
     [-modules module_list | -lib_cells lib_cell_list]
```

Specifies the action to be taken on the selected targets during simulation when the power is switched off or restored.

You can specify this command in a block-level CPF. It is equivalent to a command at the top level with the `-instances` or `-domain` options to restrict the target selection.

You must specify a list of targets for a specific action with a given type.

You can combine either the `-modules` or `-lib_cells` option with the `-instances` and `-domains` options to filter the list of targets selected with the `-target` and `-type` options.

- When you do not specify any of these options, targets are selected only within the current CPF scope.
- When you specify only the `-modules` option, a search is done for all instances of the specified module(s) through the entire hierarchy of the current CPF scope. Targets are only selected from these instances.
- When you specify only the `-instances` option, targets are only selected from the specified instances.
- When you specify only the `-domains` option, a search is done for all instances, that belong to the specified power domains, through the entire hierarchy of the current CPF scope. Targets are only selected from these instances.
- When you specify both the `-domains` and `-modules` options, targets are selected only from the instances that satisfy the search results of both options.
- When you specify both the `-instances` and `-modules` options, a search is done for all instances of the specified module(s) within the instances specifies with the `-instances` option.
- When you specify only the `-lib_cells` option, a search is done for all instances of the specified cells through the entire hierarchy of the current CPF scope. Targets are only selected from these instances.
- When you specify both the `-domains` and `-lib_cells` options, targets are selected only from the instances that satisfy the search results of both options.

- When you specify both the `-instances` and `-lib_cells` options, targets are selected only from the instances that satisfy the search results of both options.

You can use the `-exclude` option to further filter the selected targets.

To control the action of initial blocks when the power is restored from shutoff, use the `-action power_up_replay` to specify such initial statements.

Note: Initial blocks are non-synthesizable code used in simulation to create proper startup conditions at time zero of the simulation.

Options and Arguments

`-action {power_up_replay | disable_corruption | disable_isolation | disable_retention}`

Specifies the action to be taken during simulation. This option can have the following values:

- `disable_corruption`—specifies that the simulator must ignore the default simulation corruption semantics applicable to the selected targets.
- `disable_isolation`—specifies that the simulator must ignore the virtual isolation logic inferred from the selected rules during simulation.
- `disable_retention`—specifies that the simulator must ignore the virtual state retention logic inferred from the selected rules during simulation.
- `power_up_replay`—specifies that the selected initial blocks will be replayed immediately after power is restored to the domains that contain the initial blocks.

  By default, initial blocks are ignored when the domain in which they are located is powered up.

Note: For a design with mixed RTL and netlist, you can use the `disable_isolation` and `disable_retention` options to disable the CPF simulation semantics applied to the netlist in which the isolation and retention logic is already inserted, but still enable the CPF simulation semantics on other RTL blocks.

`-controlling_domain` *`domain`*

Specifies the power domain that controls the action on the selected initial blocks.

For example, for initial block replay, the replay occurs when the specified domain changes from power-down to power-up state.

Default: The default controlling domain is the power domain of the logic hierarchy containing the selected initial block.

Note: This option can only be used with the `power_up_replay` action.

`-domains` *`domain_list`*

Specifies the domains from which the targets must be selected.

Domain names can contain wildcards.

The power domains must have been previously defined with the `create_power_domain` command.

`-exclude` *`target_list`*

Excludes the specified list of targets from the list of already selected targets.

The targets can contain wildcards.

Note: Any target specified with this option will be ignored if is not in the list of the selected targets.

`-instances` *`instance_list`*

Specifies the list of hierarchical instances from which the targets must be selected.

Instance names can contain wildcards.

`-lib_cells` *`lib_cell_list`*

Specifies a list of cells from which the targets must be selected.

The selected scope is either the current CPF scope or the scope specified with the `-instances` or the `-domains` options.

Cell names can contain wildcards.

Note: The specified names cannot be specified with the `-modules` option.

`-modules` *`module_list`*

Specifies a list of modules.Targets are selected from all instances of these modules within the selected scope.

The selected scope is either the current CPF scope or the scope specified with the `-instances` or the `-domains` options.

Module names can contain wildcards.

`-targets` *`target_list`*

Specifies a list of object names in the selected scope.

- If the action is either `disable_isolation`, or `disable_retention`, the targets are rules.
- If the action is `power_up_replay`, the targets are initial blocks.
- If you omit this option, all targets of the type identified by the action are selected.

Specify the full hierarchical path for the object name. For example, `a.b.c.thisBlock`. If the type is an initial block, it is referenced by the label of the statement inside.

Note: You can use wildcards. If you use wildcards and you specify `-action power_up_replay`, all initial blocks (even the ones without labels) in the specified scope are selected.

`-type {real | integer | reg | module | instance}`

Specifies the type of the target. The type depends on the specified action.

- `real`— selects only real variables from RTL
- `wreal`— selects only real variables from RTL
- `integer`—selects only integer variables from RTL
- `reg`—selects only ‘reg’ type variables from RTL

  When you specify either the `integer` or `reg` type, you can only use the command in macro models. Otherwise it is an error. It is also an error if the selected variable is also covered by a state retention rule.

- `module`—selects instances of the specified modules
- `instance`—selects the specified instances

  When you specify the `module` (`instance`) type, the `-modules` (`-instances`) option is not allowed.

Using the `disable_corruption` action with type `module` and `instance` may cause mismatch between simulation verification and implementation.

When used with the `disable_corruption` action, these two options can only be used for simulation models that already include the power-aware functionality to disable default corruption semantics. This is useful in cases where the design has a mixture of power-aware and non power-aware simulation models.

Examples

- The following command selects all initial blocks found in all instances of the current scope whose label starts with `L`, but excludes those blocks whose label starts with `L1`.

  ```
  set_sim_control -target {L*} -exclude {L1*} -action power_up_replay
  ```

- The following command selects all initial blocks inside the instances that belong to domains `PD1` and `PD2`.

  ```
  set_sim_control -domains {PD1 PD2} -action power_up_replay
  ```

- The following command selects all virtual isolation logic inferred from the CPF rules inside the specified domains `PD1` and `PD2`.

  ```
  set_sim_control -domains {PD1 PD2} -action disable_isolation
  ```

- The following command selects all initial blocks with label `L1` from all instances of modules `M1` and `M2` starting at the current scope.

  ```
  set_sim_control -target L1 -modules {M1 M2} -action power_up_replay
  ```

- The following command selects all initial blocks with label `L` in instance `A.B.C`.

  ```
  set_sim_control -target L -instances {A.B.C} -action power_up_replay
  ```

- Assume module `foo` has an initial statement with label `thisBlock` and hierarchy `x` with module `foo` also has an initial statement with that same label. Further assume this module is instantiated in the design `top` as instances `A/A2` and `B`, and current scope is top.

```
top
    A
        A1
        A2 (foo)
    B (foo)
```

  To select the target `thisBlock` in both `A.A2` and `B`, specify:

```
set_sim_control -module foo -target thisBlock -action power_up_replay
```

  To select the target x.`thisBlock` in both `A.A2` and `B`, specify:

```
set_sim_control -modules foo -targets x.thisBlock -action power_up_replay
```

  To select the target `thisBlock` and `x.thisBlock` in both `A.A2` and `B`, specify:

```
set_sim_control -modules foo -targets {thisBlock x.thisBlock} \
-action power_up_replay
```

  To only select the target `thisBlock` in instance `A.A2`, specify:

```
set_sim_control -instances A.A2 -targets thisBlock -action power_up_replay
```

- The following examples show the block-level CPF command followed by their equivalent top-level command. The block-level command applies to block `foo`.

  - The following commands select all initial blocks with label `initial1`.

```
set_sim_control -target initial1
set_sim_control -target initial1 -instances foo
```

  - The following commands select all initial blocks with label `initial1` in domain `X`. Assume that block-level domain X cannot be mapped to any top-level domain.

```
set_sim_control -target initial1 -domain X
set_sim_control -target initial1 -domain foo/X
```

  - The following commands select all initial blocks with label `initial1` in the specified instances.

```
set_sim_control -target initial1 -instances {i1 i2 ...}
set_sim_control -target initial1 -instances {foo/i1 foo/i2 ...}
```

set_switching_activity

```
set_switching_activity
     {-all | -pins pin_list | -instances instance_list [-hierarchical]}
     { -probability float -toggle_rate float
     | [-clock_pins pin_list] -toggle_percentage float }
     [-mode mode]
```

Specifies activity values (toggle rate and probability) for the specified pins.

The toggle rate is the average number of toggle counts per time unit of a net during a given simulation time.

The probability is the probability of a net being high during a given simulation time.

Options and Arguments

`-all` Indicates to apply the specified activity values to all pins.

`-clock_pins` *pin_list*

Indicates to apply the specified activity values only to data signals associated with the specified clock pins.

`-hierarchical` Indicates to traverse the hierarchy of all specified hierarchical instances to apply the specified activity values to the outputs of all leaf instances in the hierarchy.

`-instances` *instance_list*

Indicates to apply the specified activity values to all outputs if the specified instances are non-hierarchical instances.

For hierarchical instances, it indicates to apply the specified activity values to the outputs of the leaf instances in the specified hierarchical instances (without traversing the hierarchy).

`-mode` *mode* Specifies the mode to which these values apply.

If this option is not specified, the specified value applies to all modes for which no specific values were specified.

`-pins` *pin_list* Indicates to apply the specified activity values to the specified pins.

`-probability` *`float`*

Specifies the probability value.

The probability is a floating value between `0` and `1`.

`-toggle_percentage` *`float`*

Specifies to compute the toggle rate as the multiplication of the specified value and the toggle rate of the related clock. If multiple clocks are related to the specified data pin, the clock with the worst frequency is used. If no clock is related to the data pin, the worst clock of the design is used.

The value must be a float between 0 and 100.

If you specify clock pins through the `-clock_pins` option, the computed toggle rate is only applied to the data pins related to those clock pins.

If you did not specify any clock pins, a computed toggle rate is applied to all data pins and the value for each data pin will be based on its related clock pin.

`-toggle_rate` *`float`*

Specifies the number of toggles per time unit.

A value of 0.02 using a time unit of ns, indicates that the pin toggles 0.02 times per nanosecond or at a frequency of 10MHz.

set_time_unit

```
set_time_unit
     [ns|us|ms]
```

Specifies the unit for all time values in the CPF file.

The command returns the new setting or the current setting in case the command was specified without an argument.

Options and Arguments

`[ns|us|ms]`	Specifies the time unit. You can specify any of these three values. *Default*: `ns`

Related Information

Information Inheritance on page 120

set_wire_feedthrough_ports

`set_wire_feedthrough_ports` *`port_list`*

Specifies a list of input ports and output ports of a macro cell that are internally connected to each other by a physical wire only.

These ports are excluded from any power domain defined for the macro cell.

Options and Arguments

`port_list`	Specifies the ports in the macro that are connected by a wire only. The list can contain wildcards.

Related Information

end_macro_model on page 197

set_floating_ports on page 221

set_input_voltage_tolerance on page 223

set_macro_model on page 232

update_design

```
update_design
     -name power_design
```

Appends the power intent specified between this command and a matching `end_design` to the previously declared power design identified by the `power_design_name`.

An example use for `update_design` is an existing CPF file that is clean and validated, but later on requires additional power intent for new logic added as a result of DFT.

The following commands are **not** allowed between `update_design` and `end_design`:

- `set_design`
- `set_instance`
- `set_macro_model`
- `end_design`
- `end_macro_model`
- `update_design`
- any commands that are only allowed for macro models

Options and Arguments

`-name power_design`

Specifies the name of a previously declared power design.

Related Information

end_design on page 195

set_design on page 212

update_isolation_rules

```
update_isolation_rules -names rule_list
     { -location {from | to | parent | any}
     | -within_hierarchy instance
     | -cells cell_list [-use_model -pin_mapping pin_mapping_list
          [-domain_mapping domain_mapping_list] ]
     | -prefix string
     | -suffix string
     | -open_source_pins_only}...
```

Appends the specified isolation rules with implementation information.

Note: You must specify at least one of the options besides `-names`, but you can also combine several options.

Tip

This command is only needed if you have special implementation requirements for the specified rules. If this command is not specified, the tools will automatically determine the correct cell and location according to the specifications in the `define_isolation_cell` command.

Options and Arguments

`-cells` *cell_list*

Specifies the names of the library cells that must be used as isolation cells for the selected pins.

By default, the appropriate isolation cells are chosen from the isolation cells defined with the `define_isolation_cell` command.

It is an error if the function of a specified cell conflicts with the expected isolation output specified by the `-isolation_output` option in the `create_isolation_rule` command.

Note: If the isolation rule was specified with the `-no_condition` option, the specified cells must have been defined with the `-no_enable` option.

`-domain_mapping` *`domain_mapping_list`*

Specifies the mapping of the domains in the macro model to the top-level domains.

This option can be used if there is a CPF macro model with the same name as the first name specified in `-cells`.

Use the following format to specify a domain mapping:

{*`domain_in_child_scope domain_in_parent_level_scope`*}

The specified domain mapping applies to all instantiations of the specified isolation cell.

If the macro model has multiple power domains defined, this option must be used.

The power domains must have been previously defined with the `create_power_domain` command.

`-location {to|from|parent|any}`

Specifies where to insert the isolation logic.

- `any`–indicates that the isolation logic can be inserted in any power domain to which the specified logic hierarchy belongs. This option can only be used if you specified the `-within_hierarchy` option.
- `from`—inserts the isolation logic inside a hierarchy belonging to the originating power domain. Unless further constrained, the location will be the outermost logic hierarchy of the originating power domain.
- `parent`—inserts the isolation logic in the parent hierarchy as follows:
  - If the rule is specified without the `-to` option, the parent hierarchy corresponds to the top-most logic hierarchy of the originating power domain.
  - If the rule is specified with the `-to` option, the parent hierarchy corresponds to the top-most logic hierarchy of the destination power domain

  Note: This is the only value allowed if the original rule was specified with the `-force` option.

- `to`—inserts the isolation logic inside a hierarchy belonging to the destination power domain. Unless further constrained, the location will be the outermost logic hierarchy of the destination power domain.

Default:

- `to`–if the original rule was specified without the `-force` option
- *`the hierarchical instance`* to which the pin belongs—if the original rule was specified with the `-force` option

  Note: If the hierarchical instance is a macro model or blackbox, the isolation logic is inserted in the parent hierarchy of that instance.

If you specify this option without the `-cells` option, the implementation tools can only use isolation cells whose valid location (specified through the `-valid_location` option in the `define_isolation_cell` command) matches or is compatible with the location specified through `-location`.

If you specify this option with the `-cells` option, the `-valid_location` value of the cells specified in the `define_isolation_cell` command must match (or be compatible with) the value of this option. Otherwise an error will be given.

If the original rule was specified with the `-force` option,

`-names` *`rule_list`*

Specifies the names of the rules to be updated.

The name can contain wildcards.

The rule must have been previously defined with the `create_isolation_rule` command.

`-open_source_pins_only`

Limits the pins to be isolated to the open source pins that belong to a power domain that is switched off while the driver domain remains powered on.

This implies that only those rules that were created with the `-isolation_target` option set to `to` can be updated.

`-pin_mapping` *`pin_mapping_list`*

Specifies a list of pin mappings for a complicated isolation cell that cannot be modelled using the `define_isolation_cell` command. The cell refers to the first cell specified by `-cells`.

Use the following format to specify a pin mapping:

{*`cell_pin design_pin_reference`*}

- *`cell_pin`* is the name of the pin in the cell definition
- *`design_pin_reference`* is one of the following:
  - the name of a design port or instance pin—You can prepend the "!" character to the name if the inverse of the port/pin is used to drive the cell pin
  - *`isolation_signal`*, which refers to the expression in `-isolation_condition`

The pin mapping cannot refer to the data input port or the data output port of the cell.

The cell data input port that is not specified in the pin mapping must be connected to the signal being isolated.

The cell data output port that is not specified in the pin mapping must be connected to the logic that is driven by the signal being isolated.

`-prefix` *`string`*

Specifies the prefix to be used when creating the isolation logic.

Default: `CPF_ISO_`

`-suffix` *`string`*

Specifies the suffix to be used when creating the isolation logic.

`-use_model`

Indicates that a simulation tool must use the functional model of the first cell specified in the `-cells` option.

When this option omitted, a simulation tool can apply the default isolation logic based on the `create_isolation_rule` semantics.

`-within_hierarchy` *`instance`*

Specifies to insert the isolation logic (with or without wrapper) in the specified instance.

Use a single hierarchical separator if the logic must be inserted at the top of the current scope.

If you specify this option with the `-cells` option, the `-valid_location` of the cells specified in the `define_isolation_cell` command must be compatible with the power domain of the hierarchical instance specified with this option. Otherwise an error will be given.

Note: The power domain of the specified instance takes precedence over the power domain of the selected location.

Note: You cannot use this option if the original rule was specified with the `-force` option.

Related Information

Isolation Cell on page 19

Isolation Rules on page 96

Isolation insertion on page 99

create_isolation_rule on page 145

define_isolation_cell on page 288

identify_power_logic on page 205

update_level_shifter_rules

```
update_level_shifter_rules
     -names rule_list
     { -location {from | to | parent | any}
     | -through power_domain_list
     | -within_hierarchy instance
     | -cells {cell_list | list_of_cell_lists}
     | -prefix string
     | -suffix string}...
```

Appends the specified level shifter rule with implementation information.

Note: You must specify at least one of the options besides `-names`, but you can also combine several options.

Tip

This command is only needed if you have special requirements for implementation of the specified rules. If this command is not specified, the tools will automatically determine the correct cell and location according to the specifications in the `define_level_shifter_cell` command.

Options and Arguments

`-cells {cell_list | list_of_cell_list}`

Specifies the names of the library cells to be used to bridge the specified power domains.

- If single-stage level shifters must be used, you can specify a single cell or one list of cells.
- If multi-stage level shifters are required, you can specify one cell or a list of N cell lists.
  - If a single cell is specified, it must have been defined with the `-multi_stage` option in the `define_level_shifter_cell` command.
  - Otherwise, you must specify an ordered list of N cell lists, where N is the number of stages in the level-shifting. Each list contains the cells appropriate for its stage.

By default, the appropriate level shifter cells are chosen from the cells defined with the `define_level_shifter_cell` command.

`-location {to|from|parent|any}`

Specifies where to insert the single-stage level shifters.

- `any`–indicates that the level shifter can be inserted in any power domain to which the specified logic hierarchy belongs. This option can only be used if you specified the `-within_hierarchy` option.
- `from`—inserts the level shifter inside a hierarchy belonging to the originating power domain. Unless further constrained, the location will be the outermost logic hierarchy of the originating power domain.
- `parent`—inserts the level shifter in the logic hierarchy as follows:
  - If the rule is specified without the `-to` option, the parent hierarchy corresponds to the top-most logic hierarchy of the originating power domain.
  - If the rule is specified with the `-to` option, the parent hierarchy corresponds to the top-most logic hierarchy of the destination power domain

  Note: This is the only value allowed if the original rule was specified with the `-force` option.
- `to`—inserts the level shifter inside a hierarchy belonging to the destination power domain. Unless further constrained, the location will be the outermost logic hierarchy of the destination power domain.

Default:

- `to`–if the original rule was specified without the `-force` option
- *`the hierarchical instance`* to which the pin belongs—if the original rule was specified with the `-force` option

If you specify this option without the `-cells` option, the implementation tools can only use level shifter cells whose valid location (specified through the `-valid_location` option in the `define_level_shifter_cell` command) matches or is compatible with the location specified through `-location`.

If you specify this option with the `-cells` option, the `-valid_location` of the cells specified in the `define_level_shifter_cell` command must match (or be compatible with) the value of this option. Otherwise an error will be given.

Important

This option is ignored in case multi-stage level shifters are required.

`-names` *`rule_list`*

Specifies the names of the level shifter rules to be updated.

The name can contain wildcards.

The rule must have been previously defined with the `create_level_shifter_rule` command.

`-prefix` *`string`* Specifies the prefix to be used when creating this logic.

Default: `CPF_LS_`

`-suffix` *`string`* Specifies the suffix to be used when creating this logic.

`-through` *`power_domain_list`*

Specifies that the level shifting must occur in multiple stages and be placed in the order of the specified domains.

The first stage is from a domain specified with the `-from` option in the corresponding `create_level_shifter_rule` command to the first domain specified with the `-through` option.

The last stage is from the last domain specified with the `-through` option to a domain specified with the `-to` option in the corresponding `create_level_shifter_rule` command.

The other stages are between the domains specified with the `-through` option in the order that they are specified.

Note: You can only specify this option when both the `-from` and `-to` options were specified the corresponding `create_level_shifter_rule` command. In addition, only one of the two options can contain a power domain list.

The power domains must have been previously defined with the `create_power_domain` command.

`-within_hierarchy` *`instance`*

Specifies to insert the level shifters (with or without wrapper) in the specified instance.

Use a single hierarchical separator if the level shifters are to be inserted at the top of the current scope.

If you specify this option with the `-cells` option, the `-valid_location` of the cells specified in the `define_level_shifter_cell` command must be compatible with the power domain of the hierarchical instance specified with this option. Otherwise an error will be given.

Note: The power domain of the specified instance takes precedence over the power domain of the selected location.

Note: You cannot use this option if the original rule was specified with the `-force` option.

Important

This option is not allowed with multi-stage level-shifting.

Related Information

Level Shifter Cell on page 19

Level Shifter Insertion on page 94

Modeling Level Shifters in the *Common Power Format User Guide*

create_level_shifter_rule on page 150

define_level_shifter_cell on page 293

update_nominal_condition

```
update_nominal_condition
     -name condition
     -library_set library_set [-power_library_set library_set]
```

Associates a library set with the specified nominal operating condition.

Options and Arguments

`-library_set` *`library_set`*

References the library set to be associated with the specified condition.

The library set must have been previously defined with the `define_library_set` command.

Note: This library set is used for timing analysis and optimization. When the `-power_library_set` option is not specified, this library set is also used for power analysis and optimization.

`-name` *`condition`*

Specifies the name of the nominal operating condition.

Note: The specified string cannot contain wildcards.

The condition must have been previously defined with the `create_nominal_condition` command.

`-power_library_set` *`library_set`*

References the power library set to be associated with the specified condition.

The library set must have been previously defined with the `define_library_set` command.

Note: If this option is not specified, the timing libraries specified with the `-library_set` option will be used for power analysis and optimization.

Related Information

create_nominal_condition on page 159

define_library_set on page 194

update_power_domain

```
update_power_domain
     -name domain
     [-instances instance_list ] [-boundary_ports port_list ]
     { -primary_power_net net | -primary_ground_net net
     | -equivalent_power_nets power_net_list
     | -equivalent_ground_nets ground_net_list
     | -pmos_bias_net net | -nmos_bias_net net
     | -deep_nwell_net net | -deep_pwell_net net
     | -user_attributes string_list
     | -transition_slope [float:]float |
     | -transition_latency {from_nom latency_list}
     | -transition_cycles {from_nom cycle_list clock_pin} } ...
```

Specifies implementation aspects of the specified power domain.

If a power domain is implemented in multiple disjoint physical regions, each region has its own power and ground nets. Use the `-equivalent_power_nets` and `-equivalent_ground_nets` options to declare these power and ground nets equivalent to the primary power and ground nets. In this case, the following applies:

- The primary power and ground pins of all instances in a power domain will be connected to the primary, or equivalent power and ground nets of the domain.
- A net referenced in an `-equivalent_xx_nets` option cannot be declared equivalent in more than one power domain unless those power domains have been mapped.
- If domain X is mapped into domain Y, the equivalent nets of domain X become equivalent nets in domain Y.

Note: You must specify at least one of the options besides `-name`, but you can also combine several options.

Important

A domain can be updated multiple times with `-transition_latency` and `-transition_cycles` without overwriting the previous command. In this case, all *unique* transitions (with different starting and ending states or nominal conditions) will be appended to form a complete state transition table for this domain. If any specific transition is specified more than once, the transition time or cycle specified in the last `update_power_domain` command will be used.

Options and Arguments

`-boundary_ports` *`pin_list`*

Incrementally updates a power domain's boundary port membership with the specified list. The effect of this option is the same as the `-boundary_ports` option in the `create_power_domain` command.

You can use this option to assign boundary ports created during implementation to the correct power domain.

`-deep_nwell_net` *`net`*

Specifies the net to be used for the deep nwell connection.

The net must have been previously defined with the `create_bias_net` command.

`-deep_pwell_net` *`net`*

Specifies the net to be used for the deep pwell connection.

The net must have been previously defined with the `create_bias_net` command.

`-equivalent_ground_nets` *`ground_net_list`*

Specifies a set of ground nets that are equivalent to the primary ground net of the power domain.

You can use wildcards (*) to specify a list of ground nets.

For macro cell power domains, this option specifies the boundary ports for the equivalent ground nets in the macro cell.

`-equivalent_power_nets` *`power_net_list`*

Specifies a set of power nets that are equivalent to the primary power net of the power domain.

You can use wildcards (*) to specify a list of power nets.

For macro cell power domains, this option specifies the boundary ports for the equivalent power nets in the macro cell.

`-instances` *`instance_list`*

Incrementally updates a power domain's instance membership with the specified list. The effect of this option is the same as the `-instances` option in the `create_power_domain` command.

You can use this option for correct power domain assignment of instances created during implementation.

For example, top-level buffering during physical implementation can introduce buffers in the top module; however, the power domain assignment of these buffers should be the power domain of the driving or receiving logic, not the power domain of the top module.

`-name` *`domain`*

Specifies the name of the power domain.

The power domain must have been previously defined with the `create_power_domain` command.

`-nmos_bias_net` *`net`*

Specifies the net to be used to body bias the n-type transistors of all functional gates in this power domain.

You must have declared this net using the `create_bias_net`, `create_power_nets` or `create_ground_nets` command, except when it is used in macro model. In a macro model, you can specify a cell port with this option to indicate that the net connected to this port is used as the bias net for this domain internally.

`-pmos_bias_net` *`net`*

Specifies the net to be used to body bias the p-type transistors of all functional gates in this power domain.

You must have declared this net using the `create_bias_net`, `create_power_nets` or `create_ground_nets` command, except when it is used in macro model. In a macro model, you can specify a cell port with this option to indicate that the net connected to this port is used as the bias net for this domain internally.

`-primary_ground_net` *net*

Specifies the primary ground net for all functional gates in the specified power domain.

You must have declared this net using the `create_ground_nets` command, except when it is used in a macro model.

For macro cell power domains, this option specifies the boundary port for the primary ground net in the macro cell.

`-primary_power_net` *net*

Specifies the primary power net for all functional gates in the specified power domain.

You must have declared this net using the `create_power_nets` command, except when it is used in a macro model

For macro cell power domains, this option specifies the boundary port for the primary power net in the macro cell.

`-transition_cycles {`*from_nom cycle_list clock_pin*`}`

Specifies the nominal condition of the starting power state, followed by a list of transition cycles to complete the transition to the next power state, followed by the clock pin.

Use the following format to specify the *cycle_list*:

to_nom`@[`*integer*`:]`*integer*
`[`*to_nom*`@[`*integer*`:]`*integer*`]...`

where *to_nom* is the nominal condition of the next power state.

If two numbers are specified, the first number indicates the minimum number of clock cycles needed to complete the transition, while the second number indicates the maximum number of cycles needed to complete the transition.

Note: If you specify only one value, it is considered to be the maximum number of cycles.

clock_pin specifies the name of the clock pin whose clock cycle is used to determine the power state transition time.

`-transition_latency {`*`from_nom latency_list`*`}`

Specifies the nominal condition of the starting power state, followed by a list of transition times to complete a transition to the next power state.

Use the following format to specify the *`latency_list`*:

`to_nom``@[`*`float`*`:]`*`float`*
`[`*`to_nom`*`@[`*`float`*`:]`*`float`*`]...`

where *`to_nom`* is the nominal condition of the next power state.

If two numbers are specified, the first number indicates the minimum time needed to complete the transition, while the second number indicates the maximum time needed to complete the transition.

Note: If you specify only one value, it is considered to be the maximum time.

Specify the time in the units specified by the `set_time_unit` command.

`-transition_slope [`*`float`*`:]`*`float`*

Specifies the transition rate(s) for the supply voltage of the domain during any state transition of the domain.

Note: Supply refers to the power supply, ground supply, and the body bias supply. If a domain state transition involves a change in multiple supplies, the voltage used to compute the transition time is in the following order: power, ground, body bias.

If two numbers are specified, the first number indicates the minimum transition rate, while the second number indicates the maximum transition rate.

Note: If you specify only one value, it is considered to be the maximum slope.

The unit for a transition rate is volt per time unit (specified by the `set_time_unit` command).

`-user_attributes` *`string_list`*

Attaches a list of user-defined attributes to the domain. Specify a list of strings.

Example

- A CPF file can contain the following commands:

```
update_power_domain -name PD1 -primary_power_net VDD1
update_power_domain -name PD1 -equivalent_power_nets VDD2
update_power_domain -name PD1 -primary_ground_net VSS
...
update_power_domain -name PD2 -primary_power_net VDD3 -primary_ground_net VSS
```

- The following CPF will cause an error because the primary power net `VDD1` was added to the list of equivalent power nets, instead of being specified through the `-primary_power_net` option.

```
update_power_domain -name PD1 -equivalent_power_nets {VDD1 VDD2}
update_power_domain -name PD1 -primary_ground_net VSS
...
update_power_domain -name PD2 -primary_power_net VDD3 -primary_ground_net VSS
```

Related Information

create_bias_net on page 138

create_ground_nets on page 143

create_power_domain on page 168

create_power_nets on page 182

update_power_mode

```
update_power_mode
     -name mode
     { -activity_file file -activity_file_weight weight
     | { -sdc_files sdc_file_list
       | -setup_sdc_files sdc_file_list
       | -hold_sdc_files sdc_file_list
       | -setup_sdc_files sdc_file_list -hold_sdc_files sdc_file_list}
     | -peak_ir_drop_limit domain_voltage_list
     | -average_ir_drop_limit domain_voltage_list
     | -leakage_power_limit float
     | -dynamic_power_limit float}...
```

Specifies the constraints for the power mode.

Note: You must specify at least one of the options besides `-name`, but you can also combine several options.

Options and Arguments

`-activity_file` *file*

Specifies the path to the activity file. Supported formats for the activity files are VCD, TCF, and SAIF.

Note: You must use the correct file extension to identify the format: `vcd`, `tcf`, `saif`. The extension is case insensitive. To indicate that the file is gzipped, use the `.gz` extension.

`-activity_file_weight` *weight*

Specifies the relative weight of the activities in this file in percentage. Use a positive floating number between 0 and 100.

To estimate the total average chip power over all modes, the activity weights are used to adjust the relative weight of each power mode.

Note: If the weights specified for the activity files for the different power modes do not add up to 100, an adjusted weight is used.

`-average_ir_drop_limit` *`domain_voltage_list`*

Specifies the maximum allowed average voltage change on a power net due to resistive effects in volt (V) for the specified power mode. This net must be the primary power net of the power domain to be considered in the specified mode.

Use the following format to specify the maximum allowed average voltage change in the domain:

`domain_name``@`*`voltage`*

Use a floating value for the voltage.

If a domain is omitted from this list, the value for the primary power net of this domain will be 0, unless you specified a value using the `-average_ir_drop_limit` option of the `create_power_nets` command.

The power domains must have been previously defined with the `create_power_domain` command.

`-dynamic_power_limit` *`float`*

Specifies the maximum allowed average dynamic power in the specified mode.

Default: 0 `mW`

`-hold_sdc_files` *`sdc_file_list`*

Specifies a list of SDC files to be used for the specified mode. Use this option if the SDC files contain only hold constraints.

You cannot combine this option with the `-sdc_files` option.

Note: File lists cannot contain wildcards.

`-leakage_power_limit` *`float`*

Specifies the maximum allowed average leakage power in the specified mode.

Default: 0 `mW`

`-name` *`mode`* Specifies the name of the mode.

`-peak_ir_drop_limit` *`domain_voltage_list`*

Specifies the maximum allowed peak voltage change on a power net due to resistive effects in volt (V) for the specified power mode. This net must be the primary power net of the power domain to be considered in the specified mode.

Use the following format to specify the maximum allowed peak voltage change in the domain:

`domain_name``@`*`voltage`*

Use a floating value for the voltage.

If a domain is omitted from this list, the value for the primary power net of this domain will be 0, unless you specified a value using the `-average_ir_drop_limit` option of the `create_power_nets` command.

The power domains must have been previously defined with the `create_power_domain` command.

`-sdc_files` *`sdc_file_list`*

Specifies a list of SDC files to be used for the specified mode. Use this option if the SDC files contain both setup and hold constraints.

You cannot combine this option with either the `-setup_sdc_files` or `-hold_sdc_files` options.

Note: File lists cannot contain wildcards.

`-setup_sdc_files` *`sdc_file_list`*

Specifies a list of SDC files to be used for the specified mode. Use this option if the SDC files contain only setup constraints.

You cannot combine this option with the `-sdc_files` option.

Note: File lists cannot contain wildcards.

Example

```
update_power_mode -name PM2 -sdc_files ../SCRIPTS/cm1.sdc \
-activity_file top_pm2.tcf -activity_file_weight 25
```

Related Information

Power Mode on page 18

Object Lists on page 25

create_power_mode on page 177

update_power_switch_rule

```
update_power_switch_rule
     -name string
     { -enable_condition_1 expression [-enable_condition_2 expression]
     | -acknowledge_receiver_1 expression [-acknowledge_receiver_2 expression]
     | -cells cell_list
     | -gate_bias_net power_net
     | -prefix string
     | -peak_ir_drop_limit float
     | -average_ir_drop_limit float }...
```

Appends the specified rules for power switch logic with implementation information.

Options and Arguments

`-acknowledge_receiver_1 (-acknowledge_receiver_2)` *`expression`*

Specifies the active value of the acknowledge receiver pin.

Use a unary expression to express the value. The expression is a Boolean function of an input pin a power controller.

When all acknowledge receiver pins are active, this signals to the power controller that the power switch network is turned on completely.

An acknowledge receiver pin is driven by the corresponding output pin of the last power switch cell in the power switch network used to implement this rule.

The output pin is specified through the `-stage_x_output` option of the `define_power_switch_cell` command.

`-average_ir_drop_limit` *`float`*

Specifies the maximum allowed average voltage change across a power switch due to resistive effects in volt (V).

Default: `0`

`-cells` *`cell_list`* Specifies the name of the library cells that can be used as power switch cells.

A power switch cell must have been defined with the `define_power_switch_cell` command.

`-enable_condition_1 (-enable_condition_2)` *`expression`*

Specifies the condition when the power switch should be enabled. The condition is a Boolean expression of one or more pins.

If only `-enable_condition_1` is specified, the expression is used as the enable signal for all enable pins of the power switch cell.

If both options are specified, the expression of the `-enable_condition_1,-enable_condition_2` will be used respectively as enable signal for the enable pin of stage 1 and stage 2 of the power switch cell.

Note: If the specified power domain has a shutoff condition, the support set of this expression must be a subset of the support set of the shut-off condition.

Default: the inversion of the expression specified for the shutoff condition of the power domain is used as the enable signal driver for the enable pin(s) of the power switch cell

Note: If any pin specified in the enable condition expression is also specified with a `set_equivalent_control_pins` command, it should be specified as the master pin. The pins that are equivalent to the master control pin will only be used for verification.

`-gate_bias_net` *`power_net`*

Specifies the power net connected to the gate bias power pin of the power switch cell.

Note: This option must only be specified if the specified power switch cell has a `-gate_bias_pin` option.

`-name` *`string`* Specifies the name of the power switch rule.

`-peak_ir_drop_limit` *`float`*

Specifies the maximum allowed peak voltage change across a power switch due to resistive effects in volt (V).

Default: `0`

`-prefix` *`string`* Specifies the prefix to be used when creating this logic.

Default: `CPF_PS_`

Examples

- In the following example, power domain `X` is made switchable through one on-chip switch that is controlled by `ena`. In addition, the external power net is made switchable through one off-chip switch controlled by `enb`.

```
create_power_nets -net VDD1 -external_shutoff_condition !enb
create_power_domain -name X -shutoff_condition {!(ena & enb)} -instances ...
update_power_domain -name X -primary_power_net VDD
create_power_switch_rule -name ps1 -domain X -external_power_net VDD1
update_power_switch_rule -name ps1 -enable_condition_1 ena
```

- In the following example, power domain `X` is made switchable through two parallel switches.

```
create_power_domain -name X -shutoff_condition {!(ena & enb)} -instances ...
update_power_domain -name X -primary_power_net VDD
create_power_switch_rule -name ps1 -domain X -external_power_net VDD1
update_power_switch_rule -name ps1 -enable_condition_1 ena
create_power_switch_rule -name ps2 -domain X -external_power_net VDD2
update_power_switch_rule -name ps2 -enable_condition_1 enb
```

- In the following example, power domain `X` is not switchable, but the voltage can be changed through two parallel switches.

```
create_power_domain -name X -instances ...
update_power_domain -name X -primary_power_net VDD
create_power_switch_rule -name ps1 -domain X -external_power_net VDD1
update_power_switch_rule -name ps1 -enable_condition_1 ena
create_power_switch_rule -name ps2 -domain X -external_power_net VDD2
update_power_switch_rule -name ps2 -enable_condition_1 enb
```

Related Information

create_ground_nets on page 143

create_power_nets on page 182

create_power_switch_rule on page 184

define_power_switch_cell on page 308

update_state_retention_rules

```
update_state_retention_rules
     -names rule_list
     { -cell_type string
     | -cells cell_list [-use_model -pin_mapping pin_mapping_list
          [-domain_mapping domain_mapping_list]]
     | -set_reset_control} ...
```

Appends the specified rules for state retention logic with implementation information.

By default, the appropriate state retention cells are chosen from the state retention cells defined with the `define_state_retention_cell` command.

If the `define_state_retention_cell` command is specified with the `-cell_type` option, the cells defined through that command can only be used by a retention rule that references that particular type through the `-cell_type` option in the `update_state_retention_rules` command.

If the `define_state_retention_cell` command is not specified with the `-cell_type` option, its cells can only be used by a retention rule that does no have the `-cell_type` option specified in the `update_state_retention_rules` command.

Tip

This command is only needed if you have special requirements for implementation of the specified rules. If this command is not specified, the tools will automatically determine the correct cell and location according to the specifications in the `define_state_retention_cell` command.

Options and Arguments

`-cells` *`cell_list`*	Specifies a list of library cells that can be used to map the sequential cells.
`-cell_type` *`string`*	Specifies the class of library cells that can be used to map the flops. **Note:** The specified class (cell type) must correspond to a cell type specified in a `define_state_retention_cell` command. The class is a user-defined name that allows to group cells into a class of retention cells with the same retention behavior.

If you specify this option with the `-cells` options, the `-cells` option takes precedence.

`-domain_mapping` *`domain_mapping_list`*

Specifies the mapping of the domains in the macro model to the top-level domains.

This option can be used if there is a CPF macro model with the same name as the first cell specified in `-cells`.

Use the following format to specify a domain mapping:

{*`domain_in_child_scope domain_in_parent_level_scope`*}

The specified domain mapping applies to all instances of the specified state retention cell.

If the macro model has multiple power domains defined, this option must be used.

The power domains must have been previously defined with the `create_power_domain` command.

`-names` *`rule_list`* Specifies the names of the rules to be updated.

The name can contain wildcards.

The rule must have been previously defined with the `create_state_retention_rule` command.

`-pin_mapping` *`pin_mapping_list`*

Specifies a list of pin mappings for a complicated state retention cell that cannot be modelled using the `define_state_retention_cell` command. The cell refers to the first cell specified by `-cells`.

Use the following format to specify a pin mapping:

{*`cell_pin design_pin_reference`*}

- *`cell_pin`* is the name of the pin in the cell definition

- *design_pin_reference* is one of the following:
  - the name of a design port or instance pin. You can prepend the "!" character to the name if the inverse of the port/pin is used to drive the cell pin
  - *save_signal*, which refers to the expression in -save_edge or -save_level option
  - *restore_signal*, which refers to the expression in -restore_edge or -restore_level option

-set_reset_control

Restricts the rule to those instances that are part of the asynchronous-set-reset-synchronizer circuit—a chain of one or more D type flip flops used to asynchronously set or reset other state elements (flip-flops or latches).

This option differentiates these state elements from other state elements in this power domain not used to supply set or reset signals.

-use_model

Indicates that a simulation tool must use the functional model of the first cell specified in -cells option.

When this option omitted, a simulation tool can apply the default state retention logic based on the create_state_retention_rule semantics.

Related Information

State Retention Cell on page 20

create_state_retention_rule on page 186

define_state_retention_cell on page 313

9

Library Cell-Related CPF Commands

define_always_on_cell

```
define_always_on_cell
     -cells cell_list [-library_set library_set]
     [ {-power_switchable LEF_power_pin
       |-ground_switchable LEF_ground_pin
       |-power_switchable LEF_power_pin -ground_switchable LEF_ground_pin}
       -power LEF_power_pin  -ground LEF_ground_pin ]
```

For applications that read .lib files—Identifies the library cells in the .lib files with more than one set of power and ground pins that can remain powered on even when the power domain they are instantiated in is powered down.

For applications that do not read library files—Allows to identify the instances of these cells in the netlist.

Note: The output of these cells is related to the non-switchable power and ground pins.

Options and Arguments

`-cells` *cell_list*

Identifies the specified cells as special cells that are always on.

`-ground` *LEF_ground_pin*

If this option is specified with the `-power_switchable` option, it indicates the `GROUND` pin of the specified cell.

If this option is specified with the `-ground_switchable` option, it indicates the `GROUND` pin in the corresponding LEF cell to which the ground that is on during power shut-off mode is applied.

`-ground_switchable` *LEF_power_pin*

Identifies the `GROUND` pin in the corresponding LEF cell to which the ground that is switched off during power shut-off mode is applied.

You can specify this option when you cut off the path from power to ground on the ground side (that is, use a footer cell).

`-library_set` *`library_set`*

References the library set to be used to search for the specified cells. Specify the library set name. All matching cells will be used.

If you omit this option, all library sets are searched and all matching cells will be used.

The libraries must have been previously defined in a `define_library_set` command.

`-power` *`LEF_power_pin`*

If this option is specified with the `-ground_switchable` option, it indicates the `POWER` pin of the specified cell.

If this option is specified with the `-power_switchable` option, it indicates the `POWER` pin in the corresponding LEF cell to which the power that is on during power shut-off mode is applied.

`-power_switchable` *`LEF_power_pin`*

Identifies the `POWER` pin in the corresponding LEF cell to which the power that is switched off during power shut-off mode is applied.

You can specify this option when you cut off the path from power to ground on the power side (that is, use a header cell).

Related Information

Always On Cell on page 19

define_global_cell

```
define_global_cell
     -cells cells [-library_set library_set]
     [-global_power pin ] [-global_ground pin ]
     [-local_power pin] [-local_ground pin]
     [-power_off_function {pullup | pulldown | hold }]
     [-isolated_pins list_of_pin_lists [-enable expression_list ] ]
```

For applications that read .lib files—Identifies the library cells in the .lib files with more than one set of power and ground pins that can remain functional even when the supplies to the local power and ground pins is switched off.

Note: By default, all input and output pins of this cell are related to the global power and ground pins.

Options and Arguments

`-cells` *cell_list*

Identifies the specified cells as global cell.

`-enable` *expression_list*

Specifies a list of simple expressions. Each simple expression describes the isolation control condition for the corresponding isolated pin list in the `-isolated_pins` option. If the internal isolation does not require a control signal, specify an empty string for that pin list. The number of elements in this list must correspond to the number of lists specified for the `-isolated_pins` option.

`-global_ground` *pin* Specifies the secondary ground pin.

`-global_power` *pin* Specifies the secondary power pin.

`-isolated_pins` *list_of_pin_lists*

Specifies a list of pin lists. Each pin list groups pins that are isolated internally with the same isolation control signal.

The pin lists can only contain input pins.

`-library_set` *`library_set`*

References the library set to be used to search for the specified cells. Specify the library set name. All matching cells will be used.

If you omit this option, all library sets are searched and all matching cells will be used.

The libraries must have been previously defined in a `define_library_set` command.

`-local_ground` *`pin`* Specifies the primary ground pin.

`-local_power` *`pin`* Specifies the primary power pin.

`-power_off_function {pullup | pulldown | hold}`

Determines the cell output when the supply to its local power or local ground is off:

- `hold`—the cell output holds the same value as the value before the supply of global power or ground is off.
- `pulldown`—the cell output is a logic low
- `pullup`—the cell output is a logic high

Example

- The following command defines cell `foo` as a global cell. The cell had three isolated pins: `pin1`, `pin2`, and `pin3`. Pins `pin1` and `pin2` have the same isolation control signal `iso1`, but `pin3` has no isolation control signal.

```
define_global_cell -cells foo \
-ioslated_pins { {pin1 pin2} {pin3}} -enable {!iso1 ""}
```

- The following command defines cell `AND2_AON` as a global cell. The cell has two power pins and performs the AND function as long as the supply connected to power pin `VDD` is not switched off.

```
define_global_cell -cells AND2_AON -local_power VDDSW \
-global_power VDD -local_ground VSS
```

define_isolation_cell

```
define_isolation_cell
     -cells cell_list [-library_set library_set]
     [-always_on_pins pin_list]
     [-aux_enables pin_list]
     [-power_switchable LEF_power_pin] [-ground_switchable LEF_ground_pin]
     [-power LEF_power_pin] [-ground LEF_ground_pin]
     [-valid_location {from | to | on | off | any}]
     { -enable pin [-clamp {high|low}]
     | -pin_groups group_list | -no_enable {high|low|hold} }
     [-non_dedicated]
```

For applications that read .lib files—Identifies the library cells in the .lib files that must be used as isolation cells.

For applications that do not read library files—Allows to identify the instances of isolation cells in the netlist.

Note: By default, the output pin of a multi-power rail isolation cell is related to the non-switchable power and ground pins. The non-enable input pin is related to the switchable power and ground pins.

Options and Arguments

`-always_on_pins` *pin_list*

Specifies a list of cell pins that are related to the non-switchable power and ground pins of the cells.

Note: A pin specified with this option, can be specified with other options as well.

`-aux_enables` *pin_list*

Specifies additional or auxiliary enable pins for the isolation cell. By default, all pins specified in this option are related to the switchable supply.

If the specified pin is related to the non-switchable supply, you must also specify the pin with the `-always_on_pins` option. The logic that drives this pin must be on when the isolation enable is asserted.

If the specified pin is related to the switchable supply, the pin must be set to the controlling value of the gate before the isolation enable is asserted or deasserted. The logic that drives this pin can be corrupted when the isolation enable (specified by the `-enable` option) is asserted

`-cells` *`cell_list`*

Identifies the specified cells as isolation cells.

The libraries loaded will be searched and all cells found will be identified.

`-clamp {high | low}` Indicates that the specified cells are isolation clamp cells.

You can only specify the `-power` option for a clamp high cell.

You can only specify the `-ground` option for a clamp low cell.

`-enable` *`pin`* Identifies the specified cell pin as the enable pin.

Note: The enable pin is related to the non-switchable power and ground pins of the cells.

`-ground` *`LEF_ground_pin`*

If this option is specified with the `-power_switchable` option, it indicates the `GROUND` pin of the specified cell.

If this option is specified with the `-ground_switchable` option, it indicates the `GROUND` pin in the corresponding LEF cell to which the ground that is on during power shut-off mode is applied.

`-ground_switchable` *`LEF_power_pin`*

Identifies the `GROUND` pin in the corresponding LEF cell to which the ground that is turned off during power shut-off mode is applied.

You can only specify this option when you cut off the path from power to ground on the ground side (that is, use a footer cell).

`-library_set` *`library_set`*

References the library set to be used to search for the specified cells. Specify the library set name. All matching cells will be used.

If you omit this option, all library sets are searched and all matching cells will be used.

The libraries must have been previously defined in a `define_library_set` command.

`-no_enable {low|high|hold}`

Specifies the following:

- The isolation cell does not have an enable pin
- The output of the cell when the supply for the switchable power (or ground) pin is powered down.

Note: If the cell can also be used as a level shifter, you must also specify the cell with a `define_level_shifter_cell` command.

`-non_dedicated` Allows to use specified cells as normal function cells.

`-pin_groups` *`group_list`*

Specifies a list of input-output paths for multi-bit cells. Each group in the list specifies one cell input pin, one cell output pin, and one optional enable pin that applies to the specified path.

Use the following format for each group in the list:

```
{ input_pin output_pin [enable_pin] }
```

An enable pin may appear in more than one group.

It is an error if the same input or output pin appears in more than one group.

`-power` *`LEF_power_pin`*

If this option is specified with the `-ground_switchable` option, it indicates the `POWER` pin of the specified cell.

If this option is specified with the `-power_switchable` option, it indicates the `POWER` pin in the corresponding LEF cell to which the power that is on during power shut-off mode is applied.

`-power_switchable` *`LEF_power_pin`*

Identifies the `POWER` pin in the corresponding LEF cell to which the power that is turned off during power shut-off mode is applied.

You can only specify this option when you cut off the path from power to ground on the power side (that is, use a header cell).

`-valid_location {to | from | on | off | any}`

Specifies the valid location of the isolation cell. Possible values are

- `from`—indicates that the cell can only be used for off to on isolation and must be inserted into a power domain compatible with the originating power domain. It typically relies on its primary power and ground pins for normal function and on its secondary power or ground pins to provide the isolation function.
- `to`—indicates that the cell must be inserted in a power domain compatible with the destination power domain since its normal and isolation functions rely on its primary power and ground pins. You can only use this type of cell for off to on isolation.
- `on`—indicates that the cell can only be inserted in the power domain that is not powered down. Its normal and isolation functions are typically provided by its primary power and ground pins. This cell can be used for off to on isolation or on to off isolation. When used for off to on isolation, it is equivalent to `to`.
- `off`—indicates that the cell can only be inserted in the power domain that is powered down. This option is redundant with the any option and will be deprecated a future release.
- `any`—indicates that the cell can be placed in any location. This cell can still provide its normal and isolation functions even when its primary power or ground is shut off, or it relies on its secondary power or ground pins for both functions. Therefore, this type of cell can be used for off to on isolation or on to off isolation.

Typically, these are isolation cells without power and (or) ground pins that connect through abutment of the cells (followpins), that is, with only a secondary power and ground pin.

This requires that the secondary power or ground pin connects to the primary power supplies of the secondary domain defined in the isolation rule.

Note: For a power-switched domain, a clamp-low isolation cell can be placed in any power domain as long as its ground supply is the same in the destination power domain. Similarly, for a ground-switched domain, a clamp-high isolation cell can be placed in any power domain as long as its power supply is the same in the destination power domain.

Default: `to`

Examples

- The following isolation cell can be placed in any location for a design that uses ground switches for power shutoff. VDD is the followpin for power connection and GVSS is the ground pin for global ground connection

```
define_isolation_cell -cells foo -power VDD -ground GVSS \
    -valid_location any
```

- The following examples illustrate the use of the `-pin_groups` option to specify multi-bit isolation cells with two paths:

```
define_isolation_cell ... -pin_groups { { x1 y1 iso1 } { x2 y2 iso2 } }
define_isolation_cell ... -pin_groups { { a1 o1 } { a2 o2} }
```

Related Information

Isolation Cell on page 19

Information Precedence on page 119

Modeling Isolation Cells in the *Common Power Format User Guide*

define_level_shifter_cell

```
define_level_shifter_cell
     -cells cell_list [-library_set library_set]
     [-always_on_pins pin_list]
     { -input_voltage_range {voltage_list | voltage_range_list}
       -output_voltage_range {voltage_list | voltage_range_list}
     | -ground_input_voltage_range {voltage_list | voltage_range_lsit}
       -ground_output_voltage_range {voltage_list | voltage_range_list}
     | -input_voltage_range {voltage_list | voltage_range_list}
       -output_voltage_range {voltage_list | voltage_range_list}
       -ground_input_voltage_range {voltage_list | voltage_range_list}
       -ground_output_voltage_range {voltage_list | voltage_range_list} }
     [-direction {up|down|bidir}]
     [-input_power_pin LEF_power_pin]
     [-output_power_pin LEF_power_pin]
     [-input_ground_pin LEF_ground_pin]
     [-output_ground_pin LEF_ground_pin]
     [-ground LEF_ground_pin] [-power LEF_power_pin]
     [-enable pin | -pin_groups { {input_pin output_pin [enable_pin] }... }]
     [-valid_location {to | from | either | any}]
     [-bypass_enable expression] [-multi_stage integer]
```

For applications that read .lib files—Identifies the library cells in the .lib files that must be used as level shifter cells.

For applications that do not read library files—Allows to identify the instances of level shifter cells in the netlist.

If you specify a list of voltages or ranges for the input supply voltage, you must also specify a list of voltages or voltage ranges for the output supply voltage. Both lists must be ordered and have the same number of elements. That is, each member in the list of input voltages (or ranges) has a corresponding member in the list of output voltages (or ranges).

Note: By default, the enable and output pins of this cell are related to the output power and output ground pins (specified through the `-output_power_pin` and `-output_ground_pin` options). The non-enable input pin is related to the input power and input ground pins (specified through the `-input_power_pin` and `-input_ground_pin` options).

Options and Arguments

`-always_on_pins` *`pin_list`*

Specifies a list of cell pins which must always be driven.

Note: A pin specified with this option, can be specified with other options as well.

`-bypass_enable` *`expression`*

Specifies the condition when to bypass the voltage shifting functionality.

When the expression evaluates to `true`, the cell does not perform voltage level shifting.

The expression must be a simple expression of the bypass enable input pin.

`-cells` *`cell_list`* Identifies any specified cell as a level shifter.

The libraries loaded will be searched and all cells found will be used.

`-direction {up | down | bidir}`

Specifies whether the level shifter can be used between a lower and higher voltage, or vice versa.

Default: `up`

`-enable` *`pin`* Identifies the pin that prevents internal floating when the power supply of the source power domain is powered down but the output voltage level power pin remains on.

Note: Cells that have this type of pin can be used for isolation purposes.

`-ground` *`LEF_ground_pin`*

Identifies the name of the `GROUND` pin in the corresponding LEF cell.

Note: This option can only be specified for level shifters that only perform power voltage shifting.

`-ground_input_voltage_range {`*`voltage_list`* | *`voltage_range_list`*`}`

Identifies either a single voltage, single voltage range, or list of voltages or ranges for the input (source) ground supply voltage that can be handled by this level shifter.

The voltage range must be specified as follows:

`lower_bound:upper_bound``[`*`:step`*`]`

Specify the lower bound, upper bound and voltage increment step, respectively. The voltage increment can be optional.

Note: This option should only be specified for ground voltage shifting.

`-ground_output_voltage_range {`*`voltage_list`* | *`voltage_range_list`*`}`

Identifies either a single voltage, single voltage range, or list of voltages or ranges for the output (destination) ground supply voltage that can be handled by this level shifter.

The voltage range must be specified as follows:

`lower_bound:upper_bound``[`*`:step`*`]`

Specify the lower bound, upper bound and voltage increment step, respectively. The voltage increment can be optional.

Note: This option should only be specified for ground voltage shifting.

`-input_ground_pin` *`LEF_ground_pin`*

Identifies the name of the `GROUND` pin in the corresponding LEF cell that must be connected to the primary ground net in the source power domain.

Note: This option is usually specified for ground voltage shifting.

`-input_power_pin` *`LEF_power_pin`*

Identifies the name of the `POWER` pin in the corresponding LEF cell that must be connected to the power net to which the voltage of the source power domain is applied.

Note: This option is usually specified for power voltage shifting.

`-input_voltage_range {`*`voltage_list`* | *`voltage_range_list`*`}`

Identifies either a single voltage, single voltage range, or list of voltages or ranges for the input (source) power supply voltage that can be handled by this level shifter.

The voltage range must be specified as follows:

`lower_bound:upper_bound``[`*`:step`*`]`

Specify the lower bound, upper bound and voltage increment step, respectively. The voltage increment can be optional.

Note: This option should only be specified for power voltage shifting.

`-library_set` *`library_set`*

References the library set to be used to search for the specified cells. Specify the library set name. All matching cells will be used.

If you omit this option, all library sets are searched and all matching cells will be used.

The libraries must have been previously defined in a `define_library_set` command.

`-multi_stage` *`integer`*

Identifies the stage of a multi-stage level shifter to which this definition (command) applies.

For a level shifter cell with N stages, N definitions must be specified for the same cell. Each definition must associate a number from 1 to N for this option. For more information, see Modeling Level Shifters.

`-output_ground_pin` *`LEF_ground_pin`*

Identifies the name of the `GROUND` pin in the corresponding LEF cell that must be connected to the primary ground net in the destination power domain.

Note: This option is usually specified for ground voltage shifting.

`-output_power_pin` *`LEF_power_pin`*

Identifies the name of the `POWER` pin in the corresponding LEF cell that must be connected to the power net to which the voltage of the destination power domain is applied.

Note: This option is usually specified for power voltage shifting.

`-output_voltage_range {`*`voltage_list | voltage_range_list`*`}`

Identifies either a single voltage, single voltage range, or list of voltages or ranges for the output (destination) power supply voltage that can be handled by this level shifter.

The voltage range must be specified as follows:

`lower_bound:upper_bound``[`*`:step`*`]`

Specify the lower bound, upper bound and voltage increment step, respectively. The voltage increment can be optional.

Note: This option should only be specified for power voltage shifting.

`-pin_groups` *`group_list`*

Specifies a list of input-output paths for multi-bit cells. Each group in the list specifies one cell input pin, one cell output pin, and one optional enable pin that applies to the specified path.

Use the following format for each group in the list:

```
{ input_pin output_pin [enable_pin] }
```

An enable pin may appear in more than one group.

It is an error if the same input or output pin appears in more than one group.

`-power` *`LEF_power_pin`*

Identifies the name of the `POWER` pin in the corresponding LEF cell.

Note: This option can only be specified for level shifters that only perform ground voltage shifting.

```
-valid_location {to | from | either | any}
```

Specifies the location of the level shifter cell. Possible values are:

- `from`—the power domain of the insertion hierarchy must be voltage compatible with the originating power domain.
- `to`—the power domain of the insertion hierarchy must be voltage compatible with the destination power domain.
- `either`—the power domain of the insertion hierarchy must be voltage compatible with the either the originating or the destination power domain.
- `any`—the power domain of the insertion hierarchy may belong to any power domain. This is the special case where the logic hierarchy specified through `-location parent` or `-within_hierarchy` does not have to be voltage compatible with either the originating power domain or the destination power domain.

  If the cell contains followpins, these pins must not be specified through either the `-input_power_pin, -output_power_pin, -input_ground_pin` or `-output_ground_pin` options.

  A power level shifter with this setting can be placed in any domain as long as the ground supplies are the same.

  A ground level shifter with this setting can be placed in any domain as long as the power supplies are the same.

  For a power and ground level shifter, which requires two definitions, the `-valid_location` must be the same in the two definitions.

power part	ground part
`any`	`from|to|either`
`from|to|either`	`any`
`any`	`any`

In the first case, the ground shifting part of the level shifter definition determines the location.

In the second case, the power shifting part of the level shifter definition determines the location.

In the third case, the cell can be placed in a domain whose power *and* ground supply are neither driving logic power and ground supply nor receiving logic power and ground supply.

Default: `to`

Examples

- The following command identifies level shifter cells with one power pin and one ground pin that perform power voltage shifting from 1.0V to 0.8V:

```
define_level_shifter_cell \
-cells LSHL* \
-input_voltage_range 1.0 -output_voltage_range 0.8 \
-direction down \
-output_power_pin VL -ground G
```

- The following command identifies level shifter cells that perform power voltage shifting from 0.8V to 1.V. In this case, the level shifter cells must have two power pins and one ground pin.

```
define_level_shifter_cell \
-cells LSLH* \
-input_voltage_range 0.8 -output_voltage_range 1.0 \
-direction up \
-input_power_pin VL -output_power_pin VH -ground G
```

- The following command identifies level shifter cells that perform both power voltage shifting from 0.8V to 1.V and ground voltage shifting from 0.5V to 0V. In this case, the level shifter cells must have two power pins and two ground pins.

```
define_level_shifter_cell \
-cells LSLH* \
-input_voltage_range 0.8 -output_voltage_range 1.0 \
-groun_input_voltage_range 0.5 -ground_output_voltage_range 0.0
-direction bidir \
-input_power_pin VL -output_power_pin VH \
-input_ground_pin GH -output_ground_pin GL
```

- The following command indicates that the level shifter can shift from 0.8 to 1.0 or from 1.0 to 1.2.

```
define_level_shifter_cell \
-cells LSHL* \
-input_voltage_range {0.8 1.0} -output_voltage_range {1.0 1.2} \
-direction up
```

- The following command indicates that the level shifter can shift from input range 0.8 to 0.9 to output range 1.0 to 1.1, or from input range 1.0 to 1.1 to output range 1.2 to 1.3.

```
define_level_shifter_cell \
-cells LSHL* \
-input_voltage_range {0.8:0.9 1.0:1.1} \
```

```
-output_voltage_range {1.0:1.1 1.2:1.3} \
-direction up
```

- The following examples illustrate the use of the `-pin_groups` option to specify multi-bit level shifter cells:

```
define_level_shifter_cell ... -pin_groups { { x1 y1 iso1 } {x2 y2 iso2 } }
define_level_shifter_cell ... -pin_groups { { a1 o1 } { a2 o2} }
```

Related Information

Level Shifter Cell on page 19

Information Precedence on page 119

create_level_shifter_rule on page 150

Modeling Level Shifters in the *Common Power Format User Guide*

define_open_source_input_pin

```
define_open_source_input_pin
     -cells cell_list -pin pin
     [-library_set library_set]
     [-type {nmos|pmos|both}]
```

Specifies a list of cells that contain open source input pins. You can use this command within a macro model to specify which input pin of the macro cell is open source.

These are input pins that must be isolated when the power supply of the driver is on, but the power supply of the cells to which the input pin belongs is shut off.

Options and Arguments

`-cells` *cell_list*

Specifies the cells to which the open source input pins belong.

`-library_set` *library_set*

References the library set to be used to search for the specified cells. Specify the library set name. All matching cells will be used.

If you omit this option, all library sets are searched and all matching cells will be used.

The libraries must have been previously defined in a `define_library_set` command.

`-pin` *pin*

Specifies the name of the open source input pin.

`-type {nmos|pmos|both}`

Indicates what kind of transistor is present at the open source for the specified pin.

If the type is

- `nmos` or `pmos`, then the open source pin is connected to an nmos or pmos device.
- `both`, then the open source pin is connected to both nmos and pmos devices, or a transmission gate.

define_pad_cell

```
define_pad_cell
     -cells cell_list -pad_pins pin_list
     [-isolated_pins list_of_pin_lists [-enable expression_list]]
     [-pin_groups pin_group_list] [-analog_pins pin_list]
```

Identifies a list of library cells in the .lib files that are simple pad cells.

Note: This can be used by tools that do not read .lib to identify a pad cell.

Options and Arguments

`-analog_pins` *`pin_list`*

Specifies the analog signal pins of the pad cell that must be connected with pins that are also declared as analog in either a `define_pad_cell` command or a `set_analog_ports` command in a macro model.

Note: Do not list analog pins that can be connected with a digital interface.

Note: Do not declare power and ground pins of a pad cell as analog pins.

`-cells` *`cell_list`*

Identifies any specified cell as pad cell.

`-enable` *`expression_list`*

Specifies a list of simple expressions. Each simple expression describes the isolation control condition for the corresponding isolated pin list in the `-isolated_pins` option. If the internal isolation does not require a control signal, specify an empty string for that pin list. The number of elements in this list must correspond to the number of lists specified for the `-isolated_pins` option.

`-isolated_pins` *`list_of_pin_lists`*

Specifies a list of pin lists. Each pin list groups pins that are isolated internally with the same isolation control signal.

The pin lists can only contain input pins.

`-pad_pins` *`pin_list`*

Specifies a list of cell pins that will be connected directly to the package or board. If the cell is a power pad cell, these pins can also be power and ground pins.

`-pin_groups` *`pin_group_list`*

Specifies a list of pin groups to define the related power and ground pins of data pins in each group.

Use the following format to specify a pin group:

`group_id``:{`*`pin pin pin...`*`}`

`group_id` specifies a unique ID for the specified pins. This ID is used by the `create_pad_rule` command to specify the mapping of each pin group to a top-level power domain. The group ID is a CPF object name.

`pin` is a power, ground or data pin that belong to this group.

Note: Data pins can only belong to one group.

Each power and ground pin must belong to at least one group.

Pins in a group without power and ground pins, or not specified in any pin group are considered to be floating pins.

If a group has more than one power (ground) pin, these power (ground) pins are considered to be equivalent for that group.

Related Information

Modeling a Pad Cell in the *Common Power Format User Guide*

create_pad_rule on page 166

define_power_clamp_cell

```
define_power_clamp_cell
     -cells cell_list
     -data pin
     -power pin [-ground pin_name]
     [-library_set library_set]
```

Specifies a list of diode cells used for power clamp control.

Options and Arguments

`-cells` *`cell_list`*

Identifies the specified cells as power clamp diode cells.

`-data` *`pin`* Specifies the cell pin that connects to the data signal.

`-ground` *`pin`* Specifies the cell pin that connects to the ground net.

`-library_set` *`library_set`*

References the library set to be used to search for the specified cells. Specify the library set name. All matching cells will be used.

If you omit this option, all library sets are searched and all matching cells will be used.

The libraries must have been previously defined in a `define_library_set` command.

`-power` *`pin_name`* Specifies the cell pin that connects to the power net.

define_power_clamp_pins

```
define_power_clamp_pins
     -cells cell_list
     -data_pins pin_list
     { [-type power] -power pin [-ground pin]
     | -type ground -ground pin [-power pin]
     | -type both -power pin -ground pin}
     [-library_set library_set]
```

Identifies a list of library cells that are either power, ground, or power and ground clamp cells, or complex cells that have input pins with built-in clamp diodes.

Options and Arguments

`-cells` *cell_list*

Identifies the specified cells as power clamp diode cells.

`-data_pins` *pin_list*

Specifies a list of cell input pins that have built-in clamp diodes.

`-ground` *pin* Specifies the cell pin that connects to the ground net.

`-library_set` *library_set*

References the library set to be used to search for the specified cells. Specify the library set name. All matching cells will be used.

If you omit this option, all library sets are searched and all matching cells will be used.

The libraries must have been previously defined in a `define_library_set` command.

`-power` *pin_name* Specifies the cell pin that connects to the power net.

`-type {power | ground | both}`

Specifies the type of clamp diode associated with the data pins.

- `both` indicates a power and ground clamp diode
- `ground` indicates a ground clamp diode
- `power` indicates a power clamp diode

Default: `power`

The type determines whether you need to specify the power pin, ground pin, or both.

define_power_switch_cell

```
define_power_switch_cell
     -cells cell_list [-library_set library_set]
     -stage_1_enable expression [-stage_1_output expression]
     [-stage_2_enable expression [-stage_2_output expression]]
     -type {footer|header}
     [ -enable_pin_bias [float:]float] [ -gate_bias_pin LEF_power_pin]
     [ -power_switchable LEF_power_pin  -power LEF_power_pin
     | -ground_switchable LEF_ground_pin -ground LEF_ground_pin ]
     [ -stage_1_on_resistance float [-stage_2_on_resistance float]]
     [ -stage_1_saturation_current float] [ -stage_2_saturation_current float]
     [ -leakage_current float ]
```

For applications that read .lib files—Identifies the library cells in the .lib files that must be used as power switch cells.

For applications that do not read library files—Allows to identify the instances of power switch cells in the netlist.

Note: The input enable and output enable pins of this cell are related to the non-switchable power and ground pins.

Note: This command is required if you use the `create_power_switch_rule` command.

Options and Arguments

`-cells` *`cell_list`*

Identifies the specified cells as power switch cells.

`-enable_pin_bias [`*`float:`*`]`*`float`*

Specifies the additional (minimum and maximum) voltage that can be applied to the enable pin of the power switch cell.

The voltage applied to the enable pin can be higher than the voltage of the power supply of the power switch. The bias voltage added to the voltage of the power supply determines the minimum and maximum voltage that can be applied to the enable pin.

Specify a positive floating value in volt (V).

Note: If you specify only one value, it is considered to be the maximum bias voltage.

`-gate_bias_pin` *`LEF_power_pin`*

Identifies a power pin in the corresponding LEF cell that provides the supply used to drive the gate input of the switch cell.

`-ground` *`LEF_ground_pin`*

Identifies the input ground pin of the corresponding LEF cell.

You can only specify this option when you cut off the path from power to ground on the ground side (that is, use a footer cell).

`-ground_switchable` *`LEF_ground_pin`*

Identifies the output ground pin in the corresponding LEF cell that must be connected to a switchable ground net.

You can only specify this option when you cut off the path from power to ground on the ground side (that is, use a footer cell).

`-leakage_current` *`float`*

Specifies the leakage current when the power switch is turned off. Specify the current in ampere (A).

`-library_set` *`library_set`*

References the library set to be used to search for the specified cells. Specify the library set name. All matching cells will be used.

If you omit this option, all library sets are searched and all matching cells will be used.

The libraries must have been previously defined in a `define_library_set` command.

`-power` *`LEF_power_pin`*

Identifies the input `POWER` pin of the corresponding LEF cell.

You can only specify this option when you cut off the path from power to ground on the power side (that is, use a header cell).

`-power_switchable` *`LEF_power_pin`*

Identifies the output power pin in the corresponding LEF cell that must be connected to a switchable power net.

You can only specify this option when you cut off the path from power to ground on the power side (that is, use a header cell).

`-stage_1_enable (-stage_2_enable)` *`expression`*

Specifies when the transistor driven by this input pin is turned on (enabled) or off.

If only stage 1 is specified, the switch is turned on when the expression for the `-stage_1_enable` option evaluates to `true`.

If both stages are specified, the switch is turned on when the expression for both enable options evaluates to true.

The expression is a function of the input pin. This pin must be an always-on pin.

`-stage_1_on_resistance (-stage_2_on_resistance)` *`float`*

Specifies the resistance of the power switch in the specified stage when the power switch is turned on. Specify the resistance in ohm.

`-stage_1_output (-stage_2_output)` *`expression`*

Specifies whether the output pin in the expression is the buffered or inverted output of the input pin specified through the corresponding -`stage_`*`x`*`_enable` option.

The pin specified in the `-stage_`*`x`*`_output` option is connected to the pin specified in the corresponding -`acknowledge_receiver_`*`x`* option of the `update_power_switch_rule` command.

Note: If an option is omitted, the corresponding acknowledge receiver pin is left unconnected.

`-stage_1_saturation_current (-stage_2_saturation_current)` *`float`*

Specifies the Id saturation current of the MOS transistor in the specified stage. Specify the current in ampere (A).

The saturation current—which can be found in the SPICE model—limits the maximum current that a power switch can support.

`-type {header|footer}`

Specifies whether the power switch cell is a header or footer cell.

Examples

- The following command defines a header power switch. The power switch has two stages. The power switch is completely on if the transistors of both stages are on. The stage 1 transistor is turned on by applying a low value to input `I1`. The output of the stage 1 transistor, `O1`, is a buffered output of input I1. The stage 2 transistor is turned on by applying a high value to input `I2`. The output of stage 2 transistor, `O2`, is the inverted value of input `I2`.

  ```
  define_power_switch_cell -cells 2stage_switch -stage_1_enable !I1 \
  -stage_1_output O1 -stage_2_enable I2 -stage_2_output !O2 -type header
  ```

- Assume that a power switch is connected to a power supply of 1.0V. The following command indicates that the enable pin of the power switch cell can be driven by signal of up to 1.2 V.

  ```
  define_power_switch_cell ... -enable_pin_bias 0:0.2
  ```

Related Information

Power Switch Cell on page 20

Expressions on page 31

Modeling Power Switch Cells in the *Common Power Format User Guide*

Information Precedence on page 119

create_power_switch_rule on page 184

define_related_power_pins

```
define_related_power_pins
     -data_pins pin_list
     -cells cell_list [-library_set library_set ]
     {-power LEF_power_pin | -ground LEF_ground_pin
     | -power LEF_power_pin -ground LEF_ground_pin }
```

Specifies the relationship between the power pins and data pins for cells that have more than one set of power and ground pins.

Note: You can also use this command to overwrite the default power pin and data pin association in special low power cells defined through the `define_xxx_cell` commands.

Options and Arguments

`-cells` *`cell_list`*

Specifies the cells for which the relationship between the power pins and data pins is defined.

`-data_pins` *`pin_list`*

Specifies a list of input or output data pins.

You can use wildcards (*) to specify a list of pin names.

`-ground` *`LEF_ground_pin`*

Specifies the `GROUND` pin of the corresponding LEF cell.

`-library_set` *`library_set`*

References the library set to be used to search for the specified cells. Specify the library set name. All matching cells will be used.

If you omit this option, all library sets are searched and all matching cells will be used.

The libraries must have been previously defined in a `define_library_set` command.

`-power` *`LEF_power_pin`*

Specifies the `POWER` pin of the corresponding LEF cell.

define_state_retention_cell

```
define_state_retention_cell
     -cells cell_list [-library_set library_set]
     [-cell_type string]
     [-always_on_pins pin_list]
     [-clock_pin pin]
     {-restore_function expression | -save_function expression
     |-restore_function expression -save_function expression}
     [-restore_check expression] [-save_check expression]
     [-retention_check expression]
     [-always_on_components component_list]
     [ {-power_switchable LEF_power_pin
       |-ground_switchable LEF_ground_pin
       |-power_switchable LEF_power_pin -ground_switchable LEF_ground_pin}
       -power LEF_power_pin  -ground LEF_ground_pin ]
```

For applications that read .lib files—Identifies the library cells in the .lib files that must be used as state retention cells.

For applications that do not read library files—Allows to identify the instances of state retention cells in the netlist.

Note: In what follows, power refers to the switchable power or ground in the cell. The non-switchable power and ground must be on for the cell to operate properly.

To model a retention cell with only one retention control, you can specify either `-save_function` or `-restore_function`.

- If you specify `-save_function`, the cell saves the current value when the expression changes from `false` to `true` and the power is on. The cell restores the saved value when the power is turned on.
- If you specify `-restore_function`, the cell saves the current value when the expression changes from `true` to `false` and the power is on. The cell restores the saved value when the expression changes from `false` to `true` and the power is on.

To model a retention cell with both save and restore control, both options must be specified. In this case, the cell saves the current value when the save expression is `true` and the power is on. The cell restores the saved value when the restore expression is `true` and the power is on.

It is an error if you use the same pin or same expressions for both the save and restore function. For example the following two commands are not correct:

```
define_state_retention_cell -cells My_Cell -restore_function pg -save_function !pg
define_state_retention_cell -cells foo -restore_function pg -save_function pg
```

Note: By default, all pins of this cell are related to the switchable power and ground pins unless otherwise specified.

Options and Arguments

`-always_on_components` *component_list*

Specifies a list of component names in the simulation model that are powered by the non-switchable power and ground pins.

Note: The option has only impact on tools that use the gate-level simulation models of state retention cells.

`-always_on_pins` *pin_list*

Specifies a list of cell pins that are related to the non-switchable power and ground pins of the cells.

Note: A pin specified with this option, can be specified with other options as well.

`-cells` *cell_list*

Identifies the specified cells as state retention cells.

The libraries loaded will be searched and all cells found will be used.

`-cell_type` *string* Specifies a user-defined name that allows to group the specified cells into a class of retention cells which all have the same retention behavior.

Note: This specification limits the group of cells that can be used to those requested through a `-cell_type` option of the `update_state_retention_rules` command.

`-clock_pin` *pin* Specifies the clock pin.

`-ground` *LEF_ground_pin*

If this option is specified with the `-power_switchable` option, it specifies the `GROUND` pin of the corresponding LEF cell.

If this option is specified with the `-ground_switchable` option, it indicates the `GROUND` pin in the corresponding LEF cell to which the ground net that is on during power shut-off mode is connected.

`-ground_switchable` *`LEF_power_pin`*

Identifies the `GROUND` pin in the corresponding LEF cell to which the ground that is turned off during power shut-off mode is applied.

You can specify this option when you cut off the path from power to ground on the ground side (that is, use a footer cell).

`-library_set` *`library_set`*

References the library set to be used to search for the specified cells. Specify the library set name. All matching cells will be used.

If you omit this option, all library sets are searched and all matching cells will be used.

The libraries must have been previously defined in a `define_library_set` command.

`-power` *`LEF_power_pin`*

If this option is specified with the `-ground_switchable` option, it indicates the `POWER` pin of the specified cell.

If this option is specified with the `-power_switchable` option, it indicates the `POWER` pin to which the power that is always on during shut-off mode is applied.

`-power_switchable` *`LEF_power_pin`*

Identifies the `POWER` pin in the corresponding LEF cell to which the power that is turned off during power shut-off mode is applied.

You can specify this option when you cut off the path from power to ground on the power side (that is, use a header cell).

`-restore_check` *`expression`*

Specifies the additional condition when the states of the sequential elements can be restored. The expression must be a function of the cell input pins. The expression must be `true` when the restore event occurs.

Note: If you want to use the clock pin in the expression, you must have identified the clock pin with the `-clock_pin` option.

`-restore_function` *`expression`*

Specifies the polarity of the restore pin that enables the retention cell to restore the saved value after exiting power shut-off mode. By default, the restore pin relates to the non-switchable power and ground pin of the cell. To overwrite this relationship, use the `define_related_power_pin` command.

Note: Expression is limited to the pin name and the inversion of the pin name. An expression containing only the pin name indicates an active high polarity. An expression containing the inversion of the pin name indicates an active low polarity.

`-retention_check` *`expr`*

Specifies an additional condition that must be met (after the primary power domain of the retention cell is shut off and before the power domain is powered on again) for the retention operation to be successful.

The expression (*`expr`*) can be a Boolean function of cell input pins.

The expression must be `true` when the primary power domain of the retention logic is shut off and the retention supply is on.

`-save_check` *`expression`*

Specifies the additional condition when the states of the sequential elements can be saved. The expression must be a function of the cell input pins. The expression must be `true` when the save event occurs.

Note: If you want to use the clock pin in the expression, you must have identified the clock pin with the `-clock_pin` option.

`-save_function` *`expression`*

Specifies the polarity of the save pin that enables the retention cell to save the current value before entering power shut-off mode. By default, the save pin relates to the non-switchable power and ground pin of the cell. To overwrite this relationship, use the `define_related_power_pin` command.

If not specified, the save event is triggered by the negation of the expression specified for the restore event.

Note: Expression is limited to the pin name and the inversion of the pin name. An expression containing only the pin name indicates an active high polarity. An expression containing the inversion of the pin name indicates an active low polarity.

Example

- In the following example, clock `clk` must be held to `0` to save or restore the state of the sequential element. If power gating pin `pg` is set to 0, the state will be saved. If restore pin `my_restore` is set to `1`, the state will be restored.

```
define_state_retention_cell -cells My_Cell -power VDDC \
-ground VSS -power_switchable VDD -restore_function "my_restore" \
-restore_check "!clk" -save_function "!pg"
```

- The following example shows how to model a master-slave type state retention cell (a state retention cell that does not have a save or restore control pin). The clock signal acts as the retention control signal.

```
define_state_retention_cell -cells ms_ret -power VDDC \
-ground VSS -power_switchable VDD -restore_check "!clk" -save_function clk
```

  This cell has no dedicated retention control pin. When the main power `VDD` is shut off, the value saved with the previous rising clock edge will be kept as long as `VDDC` is on.

Related Information

State Retention Cell on page 20

Information Precedence on page 119

Handling Master-Slave Type Retention Cells on page 108

Modeling State Retention Cells in the *Common Power Format User Guide*

A

Quick Reference

```
assert_illegal_domain_configurations -name mode
     { -domain_conditions domain_condition_list
     | -group_modes group_modes_list
     | -domain_conditions domain_condition_list -group_modes group_mode_list }

create_analysis_view
     -name string
     -mode mode
     { -domain_corners domain_corner_list
     | -group_views group_view_list
     | -domain_corners domain_corner_list -group_views group_view_list}
     [ -user_attributes string_list] [-default]

create_assertion_control
     -name string
     { -assertions assertion_list
     | -domains power_domain_list}
     [ -exclude assertion_list]
     [ -shutoff_condition expression]
     [ -type {reset | suspend} ]

create_bias_net
     -net net
     [-driver pin]
     [-user_attributes string_list]
     [-peak_ir_drop_limit float]
     [-average_ir_drop_limit float]

create_global_connection
     -net net
     { -pins pin_list | -ports port_list | -pg_type pg_type_string }
     [-domain power_domain | -instances instance_list]

create_ground_nets
     -nets net_list
     [-voltage {float | voltage_range}]
     [-external_shutoff_condition expression | -internal]
     [-user_attributes string_list]
     [-peak_ir_drop_limit float]
     [-average_ir_drop_limit float]
```

```
create_isolation_rule
     -name string
     [-isolation_condition expression | -no_condition]
     { -force -pins pin_list
     | -from power_domain_list | -to power_domain_list
     | -from power_domain_list -to power_domain_list }
     [-pins pin_list]
     [-exclude pin_list]
     [-isolation_target {from|to}]
     [-isolation_output { low | high | hold | tristate |clamp_high | clamp_low}
     [-isolation_control list_of_additional_controls]
     [-secondary_domain power_domain]

create_level_shifter_rule
     -name string
     { -force -pins pin_list
     | -from power_domain_list | -to power_domain_list
     | -from power_domain_list -to power_domain_list }
     [-pins pin_list]
     [-exclude pin_list]
     [-bypass_condition expression]
     [-output_domain power_domain] [-input_domain power_domain]

create_mode
     -name string -condition expression
     [-probability float] [-illegal]

create_mode_transition
     -name string
     -from power_mode -to power_mode
     [-assertions assertion_list]
     { -start_condition expression [-end_condition expression]
       [ -cycles [integer:]integer -clock_pin clock_pin
       | -latency [float:]float ]
     | -illegal }

create_nominal_condition
     -name string
     -voltage {voltage | voltage_list}
     [-ground_voltage {voltage | voltage_list}]
     [-state {on | off | standby}]
     [-pmos_bias_voltage {voltage | voltage_list}]
     [-nmos_bias_voltage {voltage | voltage_list}]
     [-deep_pwell_voltage {voltage | voltage_list}]
     [-deep_nwell_voltage {voltage | voltage_list}]
```

```
create_operating_corner
     -name corner
     -voltage float [-ground_voltage float]
     [-pmos_bias_voltage float] [-nmos_bias_voltage float]
     [-process float]
     [-temperature float]
     -library_set library_set_list [-power_library_set library_set_list]

create_pad_rule -name string
     {-of_bond_ports port_list | -instances instance_list}
     -mapping {mapping_list}

create_power_domain
     -name power_domain
     [-instances instance_list] [-exclude_instances instance_list]
     [-boundary_ports pin_list [-exclude_ports pin_list]] [-default]
     [-shutoff_condition expression [-external_controlled_shutoff]]
     [-default_isolation_condition expression ]
     [-default_restore_edge expr | -default_save_edge expr
     |-default_restore_edge expr -default_save_edge expr
     |-default_restore_level expr -default_save_level expr ]
     [-power_up_states {random|high|low|inverted} ]
     [-power_down_states {low|high|random|inverted}]
     [-active_state_conditions active_state_condition_list ]
     [-base_domains domain_list] [-power_source]

create_power_mode
     -name string [-default]
     {-domain_conditions domain_condition_list
     |-group_modes group_mode_list
     |-domain_conditions domain_condition_list -group_modes group_mode_list }
     [-condition expression]

create_power_nets
     -nets net_list
     [-voltage {float | voltage_range}]
     [-external_shutoff_condition expression | -internal]
     [-user_attributes string_list]
     [-peak_ir_drop_limit float]
     [-average_ir_drop_limit float]

create_power_switch_rule
     -name string
     -domain power_domain
     {-external_power_net net | -external_ground_net net}
```

```
create_state_retention_rule
     -name string
     {-domain power_domain | -instances instance_list}
     [-exclude instance_list]
     [-required]
     [ -restore_edge expr | -save_edge expr
          |-restore_edge expr -save_edge expr
          |-restore_level expr -save_level expr]
     [-restore_precondition expr] [-save_precondition expr]
     [-retention_precondition expr]
     [-target_type {flop|latch|both}]
     [-secondary_domain domain] [-use_secondary_for_output]

define_always_on_cell
     -cells cell_list [-library_set library_set]
     [ {-power_switchable LEF_power_pin
       |-ground_switchable LEF_ground_pin
       |-power_switchable LEF_power_pin -ground_switchable LEF_ground_pin}
       -power LEF_power_pin  -ground LEF_ground_pin ]

define_global_cell
     -cells cells [-library_set library_set]
     [-global_power pin ] [-global_ground pin ]
     [-local_power pin] [-local_ground pin]
     [-power_off_function {pullup | pulldown | hold }]
     [-isolated_pins list_of_pin_lists [-enable expression_list ] ]

define_isolation_cell
     -cells cell_list [-library_set library_set]
     [-always_on_pins pin_list]
     [-aux_enables pin_list]
     [-power_switchable LEF_power_pin] [-ground_switchable LEF_ground_pin]
     [-power LEF_power_pin] [-ground LEF_ground_pin]
     [-valid_location {from | to | on | off | any}]
     { -enable pin [-clamp {high|low}]
     | -pin_groups group_list | -no_enable {high|low|hold} }
     [-non_dedicated]
```

```
define_level_shifter_cell
     -cells cell_list [-library_set library_set]
     [-always_on_pins pin_list]
     { -input_voltage_range {voltage_list | voltage_range_list}
       -output_voltage_range {voltage_list | voltage_range_list}
     | -ground_input_voltage_range {voltage_list | voltage_range_lsit}
       -ground_output_voltage_range {voltage_list | voltage_range_list}
     | -input_voltage_range {voltage_list | voltage_range_list}
       -output_voltage_range {voltage_list | voltage_range_list}
       -ground_input_voltage_range {voltage_list | voltage_range_list}
       -ground_output_voltage_range {voltage_list | voltage_range_list} }
     [-direction {up|down|bidir}]
     [-input_power_pin LEF_power_pin]
     [-output_power_pin LEF_power_pin]
     [-input_ground_pin LEF_ground_pin]
     [-output_ground_pin LEF_ground_pin]
     [-ground LEF_ground_pin] [-power LEF_power_pin]
     [-enable pin | -pin_groups { {input_pin output_pin [enable_pin] }... }]
     [-valid_location {to | from | either | any}]
     [-bypass_enable expression] [-multi_stage integer]

define_library_set
     -name library_set
     -libraries list
     [-user_attributes string_list]

define_open_source_input_pin
     -cells cell_list -pin pin
     [-library_set library_set]
     [-type {nmos|pmos|both}]

define_pad_cell
     -cells cell_list -pad_pins pin_list
     [-isolated_pins list_of_pin_lists [-enable expression_list]]
     [-pin_groups pin_group_list] [-analog_pins pin_list]

define_power_clamp_cell
     -cells cell_list
     -data pin
     -power pin [-ground pin_name]
     [-library_set library_set]

define_power_clamp_pins
     -cells cell_list
     -data_pins pin_list
     { [-type power] -power pin [-ground pin]
     | -type ground -ground pin [-power pin]
     | -type both -power pin -ground pin}
     [-library_set library_set]
```

```
define_power_switch_cell
     -cells cell_list [-library_set library_set]
     -stage_1_enable expression [-stage_1_output expression]
     [-stage_2_enable expression [-stage_2_output expression]]
     -type {footer|header}
     [ -enable_pin_bias [float:]float] [ -gate_bias_pin LEF_power_pin]
     [ -power_switchable LEF_power_pin  -power LEF_power_pin
     | -ground_switchable LEF_ground_pin -ground LEF_ground_pin ]
     [ -stage_1_on_resistance float [-stage_2_on_resistance float]]
     [ -stage_1_saturation_current float] [ -stage_2_saturation_current float]
     [ -leakage_current float ]

define_related_power_pins
     -data_pins pin_list
     -cells cell_list [-library_set library_set ]
     {-power LEF_power_pin | -ground LEF_ground_pin
     | -power LEF_power_pin -ground LEF_ground_pin }

define_state_retention_cell
     -cells cell_list [-library_set library_set]
     [-cell_type string]
     [-always_on_pins pin_list]
     [-clock_pin pin]
     {-restore_function expression | -save_function expression
     |-restore_function expression -save_function expression}
     [-restore_check expression] [-save_check expression]
     [-retention_check expression]
     [-always_on_components component_list]
     [ {-power_switchable LEF_power_pin
       |-ground_switchable LEF_ground_pin
       |-power_switchable LEF_power_pin -ground_switchable LEF_ground_pin}
       -power LEF_power_pin  -ground LEF_ground_pin ]

end_design
     [power_design_name]

end_macro_model
     [macro_mode_name]

end_power_mode_control_group

find_design_objects pattern
     [-pattern_type {name | cell | module}]
     [-scope hierchical_instance_list]
     [-object {inst | port | pin | net}] [-direction {in | out | inout}]
     [-leaf_only | -non_leaf_only ]
     [-hierarchical] [-exact | -regexp] [-ignore_case]

get_parameter parameter_name

identify_always_on_driver
     -pins pin_list [-no_propagation]
```

```
identify_power_logic
     -type isolation
     {-instances instance_list | -module name}

identify_secondary_domain
     -secondary_domain domain
     {-instances instance_list | -cells cell_list }
     [ -domain power_domain [-from power_domain | -to power_domain]]

include file

set_analog_ports port_list
     [-user_attributes string_list]

set_array_naming_style
     [string]

set_cpf_version
     [value]

set_design
     power_design [-modules module_list]
     [-ports port_list] [-input_ports port_list]
     [-output_ports port_list] [-inout_ports port_list]
     [-honor_boundary_port_domain]
     [-parameters parameter_value_list]
     [-testbench] ...

set_diode_ports
     {-positive pos_port_list -negative neg_port_list
     |-positive pos_port_list | -negative neg_port_list}

set_equivalent_control_pins
     -master pin
     -pins pin_expression_list
     { -domain domain| -rules rule_list }

set_floating_ports port_list

set_hierarchy_separator
     [character]

set_input_voltage_tolerance
     { -power lower[:upper] | -ground [lower:]upper
     | -power lower[:upper] -ground [lower:]upper }
     [-domain power_domain] [-pins pin_list]

set_instance
     [ instance
       [-design power_design | -model macro_model]
       [-port_mapping port_mapping_list]
       [-domain_mapping domain_mapping_list]
       [-parameter_mapping parameter_mapping_list] ]

set_macro_model macro_model_name [-cells -cell_list]
```

```
set_pad_ports pin_list

set_power_mode_control_group -name group
     { -domains domain_list
     | -groups group_list
     | -domains domain_list -groups group_list}

set_power_source_reference_pin pin
     -domain power_domain -voltage_range min:max

set_power_target
     { -leakage float | -dynamic float
     | -leakage float -dynamic float}

set_power_unit
     [pW|nW|uW|mW|W]

set_register_naming_style
     [string%s]

set_sim_control [-targets target_list [-exclude target_list]]
     {-action power_up_replay [-controlling_domain domain ]
     |-action disable_corruption -type { real | wreal | integer | reg | module |
        instance}
     |-action {disable_isolation | disable_retention} }
     [-domains domain_list | -instances instance_list]
     [-modules module_list | -lib_cells lib_cell_list]

set_switching_activity
     {-all | -pins pin_list | -instances instance_list [-hierarchical]}
     { -probability float -toggle_rate float
     | [-clock_pins pin_list] -toggle_percentage float }
     [-mode mode]

set_time_unit
     [ns|us|ms]

set_wire_feedthrough_ports port_list

update_design
     -name power_design_name

update_isolation_rules -names rule_list
     { -location {from | to | parent | any}
     | -within_hierarchy instance
     | -cells cell_list [-use_model -pin_mapping pin_mapping_list
          [-domain_mapping domain_mapping_list] ]
     | -prefix string
     | -suffix string
     | -open_source_pins_only}...
```

```
update_level_shifter_rules
     -names rule_list
     { -location {from | to | parent | any}
     | -through power_domain_list
     | -within_hierarchy instance
     | -cells {cell_list | list_of_cell_lists}
     | -prefix string
     | -suffix string}...

update_nominal_condition
     -name condition
     -library_set library_set [-power_library_set library_set]

update_power_domain
     -name domain
     [-instances instance_list ] [-boundary_ports port_list ]
     { -primary_power_net net | -primary_ground_net net
     | -equivalent_power_nets power_net_list
     | -equivalent_ground_nets ground_net_list
     | -pmos_bias_net net | -nmos_bias_net net
     | -deep_nwell_net net | -deep_pwell_net net
     | -user_attributes string_list
     | -transition_slope [float:]float |
     | -transition_latency {from_nom latency_list}
     | -transition_cycles {from_nom cycle_list clock_pin} } ...

update_power_mode
     -name mode
     { -activity_file file -activity_file_weight weight
     | { -sdc_files sdc_file_list
       | -setup_sdc_files sdc_file_list
       | -hold_sdc_files sdc_file_list
       | -setup_sdc_files sdc_file_list -hold_sdc_files sdc_file_list}
     | -peak_ir_drop_limit domain_voltage_list
     | -average_ir_drop_limit domain_voltage_list
     | -leakage_power_limit float
     | -dynamic_power_limit float}...

update_power_switch_rule
     -name string
     { -enable_condition_1 expression [-enable_condition_2 expression]
     | -acknowledge_receiver_1 expression [-acknowledge_receiver_2 expression]
     | -cells cell_list
     | -gate_bias_net power_net
     | -prefix string
     | -peak_ir_drop_limit float
     | -average_ir_drop_limit float }...
```

```
update_state_retention_rules
     -names rule_list
     { -cell_type string
     | -cells cell_list [-use_model -pin_mapping pin_mapping_list
          [-domain_mapping domain_mapping_list]]
     | -set_reset_control} ...
```

Index

L

M

N

O

P

R

S

T

U

V

W