Si2 Common Power Format Specification™

Version 2.0

14-February-2011

Published by
Silicon Integration Initiative, Inc. (Si2^{TM})
9111 Jollyville Road, Suite 250
Austin TX 78759

ISBN: 1-882750-57-8

Document Status

This Document is a DRAFT for Si2 CPF Format Working Group Members only.

Released version: si2_cpf_v2.0_14-feb-2011.pdf

This section reflects the status of this document at the time of its publication. Other documents may supersede this document. Please contact Si2 for a complete list of current Si2 publications produced by the LPC.

Formal comments and error reports on this document should be sent to Si2 Common Power Format Project on http://www.si2.org.

The Si2 CPF Specification Version 2.0 (14 February 2011) was approved by the LPC Technical Steering Group and Low Power Coalition.

Revision History

Version	Date	Comments
2.0	14-Feb-2011	Numerous changes and improvements, including improved semantics for hierarchical flow, isolation, level shifting, state retention, modes, pad modeling, simulation, macro modeling, and others. See Enhancements and New Capabilities for CPF 2.0 on page 9
1.1	19-Sep-2008	Added new syntax and semantics for better support of hierarchy.
1.0	02-Jan-2007	Initial specification

Acknowledgements

This document has been produced by the LPC Format Working Group as part of the Low Power Coalition.

LPC Format Working Group

Nick English	Si2
Josefina Hobbs	Synopsys
Anmol Mathur	Calypto
Judith Richardson	AMD
Sheela Shreedharan	Synopsys
Prasad Subbarao	LSI
Steve Urish	IBM
Qi Wang	Cadence Design Systems

Editor

Susan Carver Si2

Contents

Preface

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.
`arguments and options`	Words in italics indicate user-defined arguments or options for which you must substitute a name or a value.
|	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` }
`...`	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.
{ }	Braces in bold-face type must be entered literally.
#	The pound sign precedes comments.

Enhancements and New Capabilities for CPF 2.0

Note: A summary list of changes on a per-section basis is in Appendix B, "Chapter by Chapter Summary of Changes" on page 301.

Design and Instance Semantic Changes

Changed Semantics for set_design

The semantics of set_design have been changed:

- CPF1.1 specified the power intent for one Module.
- CPF2.0 specifies an entity, referred to as a Power Design, that defines power intent. A power design may be applied to
  - a top design, or
  - instantiations of any module, or
  - instantiations of one or more explicitly specified modules

Appending Power Intent to an Existing Design

- CPF1.1 appends power intent to an existing design using additional set_design end_design pairs
- CPF2.0 appends power intent to an existing power design using the new update_design command paired with end_design

Expanded Semantics for Virtual Ports

- In CPF 1.1, the `-ports` option of the set_design command specified input ports only.
- In CPF 2.0, new options explicitly specify `-input_ports`, `-output_ports`, and `-inout_ports`.

  Note: The existing `-ports` option still exists for backward compatibility.

Clarified Semantics for Instance Port Mapping

The semantics of the `-port_mapping` option of the set_instance command have been clarified to include the specification of real ports in addition to virtual ports.

New Concepts

Introduced the Generic Mode

The create_mode command has been added to create a generic mode that applies to a subset of domains.

Introduced Concept of Power Source Domain

Introduced the concept of a Power Source Domain, that is a power domain that contains the output supply of a voltage regulator.

Introduced the Concept of Global Cell

The global cell replaces the always-on cell. The global cell is a special cell that has secondary power or ground pins in addition to the primary power and ground pins (followpins).

To support this new cell, the define_global_cell command was added.

Improvement to Hierarchical Flow

Changed Semantics of Port Mapping

Starting with this version, macro model pins can be considered for port mapping in the set_instance command.

Added Support for Specification of Macro Cell Ports Connected to Bonding Ports

To specify a set of macro cell ports that can directly connect to a bonding port at the chip level, the set_pad_ports command was added. No low power logic such as level shifters and isolation is allowed between the top-level bonding ports and the pad ports of a macro.

Added Support for Specification of Macro Cell Ports Connected to Diode Cells

To specify a list of pins of a macro cell that connect to the positive and negative pins of a diode, the set_diode_ports command was added.

Added Support for Assigning Leaf-level Library Cell instances to Power Domains

The `-instances` option of the create_power_domain command in CPF 1.1 included instances of standard cells and macro cells. The semantics have been expanded to allow instances of *any* library cells.

Added Capability to Create a Single Macro Model that can Apply to Multiple Cells

The `-cells` option has been added to the set_macro_model command to allow a single macro model to apply more than one cell. For example, different memory cells with the same architecture but different bits can use the same macro cell definition.

Use of this option also facilitates better error checking in cases where the user mistakenly applies the wrong macro model to an instance of a cell.

Added Capability to Find Design Objects within a Specified Scope

The new command find_design_objects locates and returns design objects which may then be used as arguments to other CPF commands.

Isolation Enhancements

Changed Semantics of Location in Rules

- In CPF 1.1, when you specified the `-location` and `-cells` options together in the update_isolation_rules (or update_level_shifter_rules) command, the `-cells` option took precedence—the location of the cells was determined by the `-valid_location` specification in the define_isolation_cell (or define_level_shifter_cell) command.

- In CPF 2.0, if you specify the `-location` and `-cells` options together in the update_isolation_rules (or update_level_shifter_rules`) command`, the value of the `-location` option must match the value specified for the `-valid_location` in the define_isolation_cell (or define_level_shifter_cell) command. Otherwise an error will be given.

Changed Semantics of -within_hierarchy in Rules

- In CPF 1.1, when you specified the `-within_hierarchy` option in the update_isolation_rules command, the power domain of the specified instance had to match the power domain of the default location or the location specified with the `-location` option.
- In CPF 2.0, the power domain of the specified instance no longer needs to match the power domain specified by the `-location` option. The power domain of the instance specified with the `-within_hierarchy` option takes precedence.

  This change accommodates users who want to put the isolation in neither the 'from' nor the 'to' power domain. This can be common due to physical constraints that require the placement of the isolation in a separate domain.

  Also, if you specify the `-within_hierarchy` and `-cells` options together, 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 the `-within_hierarchy` option in the update_isolation_rules. Otherwise an error will be given.

Added Support for Isolation Cells That Can Be Placed in Any Domain

Isolation cells without power and ground pins that connect through abutment of the cells (followpins), that is, with only a secondary power and ground pin, can be placed in any domain.

To support this type of isolation cell, the following changes were made to the define_isolation_cell command:

- Added the `any` value for the `-valid_location` option
- The command can be specified without the `-power_switchable` and `-ground_switchable` options

Implementation tools must connect the secondary power and ground pin of this isolation cell to the supplies of the secondary domain of the instance. See Secondary Power Domain of Isolation Instances on page 65.

Added Support for Isolation Cells That Can be Placed in Power and Ground Shutoff Domains

To support these types of isolation cells, the define_isolation_cell command can be specified with both the `-power_switchable` and `-ground_switchable` options.

Added Support for Isolation Logic That Can Be Placed in Parent Hierarchy

The `-location` option of update_isolation_rules now accepts the value `parent`. This supports the placement of isolation logic in the logic hierarchy of the parent instance of the nets that were selected by the `-from`, `-to`, and `-pins` options in create_isolation_rule.

Added Support for Clamp-Type Isolation Cell

To support isolation clamp-type isolation cells, the following changes were made:

- Added the `-clamp` option to the define_isolation_cell command.
- Added the `clamp_high` and `clamp_low` values for the `-isolation_output` option of the create_isolation_rule command

Added Capability to Force the Insertion of Isolation Logic

Added the `-force` option to the create_isolation_rule command.

Added Capability to Specify a Name Suffix

Added the `-suffix` option to the update_isolation_rules command.

Added Support to Enable Multi-bit Cells

Added the `-pin_group` option to the define_isolation_cell command.

Level Shifter Enhancements

Expanded Support for Input Voltage Tolerance

- In CPF 1.1, input voltage tolerance could only be specified for input pins of a macro model.
- In CPF 2.0, the concept has been generalized for all input pins. You can also specify a different tolerance for power and ground voltages. You can specify the pins or the domain to which the constraints apply. If the input voltage tolerance is violated, a power or ground level shifter is required for the specified pins.To support these new features the syntax of the set_input_voltage_tolerance command was revised.

Changed Semantics of -within_hierarchy in Rules

- In CPF 1.1, when you specified the `-within_hierarchy` option in the update_level_shifter_rules command, the power domain of the specified instance had to match the power domain of the default location or the location specified with the `-location` option.
- In CPF 2.0, the power domain of the specified instance no longer needs to match the power domain specified by the `-location` option. The power domain of the instance specified with the `-within_hierarchy` option takes precedence.

  This change accommodates users who want to put a level shifter in neither the ‘from’ nor the ‘to’ power domain. This can be common due to physical constraints that require the placement of the level shifter in a separate domain.

  Also, if you specify the `-within_hierarchy` and `-cells` options together, 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 the `-within_hierarchy` option in the update_level_shifter_rules command. Otherwise an error will be given.

Added Support for Level Shifter Cells That Can Be Placed in Any Domain

Level shifter cells whose input and output pins are specified as non-followpins can be placed in any domain.

To support this type of level shifter cell, the `any` value was added for the `-valid_location` option of the define_level_shifter_cell command.

If the valid location of a level shifter cell is `any`, it can be used in a rule using `-within_hierarchy` with a module belonging to any power domain.

Added Support for Level Shifters That Can Be Placed in Parent Hierarchy

The `-location` option of update_level_shifter_rules now accepts the value parent. This supports the placement of level shifter logic in the logic hierarchy of the parent instance of the nets that were selected by the `-from`, `-to`, and `-pins` options in create_level_shifter_rule.

Added Support for Bypass Level Shifter

To support bypass level shifters, the following changes were made:

- Added the `-bypass_enable` option to the define_level_shifter_cell command
- Added the `-bypass_condition` option to the create_level_shifter_rule command.

Added Support to Specify Lists for Input and Output Voltages in Level Shifter Definitions

- In CPF 1.1, you needed to specify either a single voltage or a voltage range for the input (source) and output (destination) supply voltages. However, when specifying ranges, some unintended input and output combination could be implied. Consider the following option combination in the define_level_shifter_cell command:

  ```
  -input_voltage_range 0.8:1.0:0.2 -output_voltage_range 1.0:1.2:0.2
  ```

  This could imply that the level shifter can shift from 0.8 to 1.2, while the level shifter can only shift from either 0.8 to 1.0 or from 1.0 to 1.2.
- In CPF 2.0, you can specify lists of voltages and voltage ranges. 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).

Added Support for Multi-Stage Level Shifters

To support multi-stage level shifters, the following changes were made:

- Added the `-multi_stage` option to the define_level_shifter_cell command.

- Added the `-through` option to the update_level_shifter_rules command, which specifies the subsequent domains for the multi-stage level shifting between the first domain (specified with `-from`) and the last domain (specified with `-to`).

  In addition, the `-cells` option of the update_level_shifter_rules command can accept a list of lists for multi-stage level shifters.

Added Capability to Force the Insertion of Level Shifter Logic

Added the new `-force` option to the create_level_shifter_rule command.

Added Capability to Specify a Name Suffix

Added the `-suffix` option to the update_level_shifter_rules command.

Added Support to Enable Multi-bit Cells

Added the `-pin_group` option to the define_level_shifter_cell command.

State Retention Enhancements

Added Support for Retention Check and Precondition

A `-retention_precondition` option was added to the create_state_retention_rule command and a `-retention_check` option to the define_state_retention_cell command.

Added Capability to Preserve Output While Power is Off

Added the `-use_secondary_for_output` option to the create_state_retention_rule command to prevent the corruption of output when the primary power is off.

Added Capability to Require the Implementation of Retention Logic

Added the `-required` option to the create_state_retention_rule command.

Power Mode and Power Mode Control Group Changes

Power Mode Control Group Semantic Change

When compound power modes are specified, a design can be in more than one power mode at a time. This can cause problems during verification when deterministic domain states are expected.

By allowing a power domain to be specified in multiple power mode control groups, this problem can be solved.

Note: In CPF 1.1, a power domain could only belong to one power mode control group.

Added Capability to Specify Illegal Power Mode Transitions

To support this feature, the `-illegal` option was added to the create_mode_transition command.

Added Capability to Specify Condition for Power Mode

In CPF 2.0, the `-condition` option was added to the create_power_mode command to indicate when the design is in the specified mode.

This option is only interpreted by simulation and verification tools.

Pad and Regulator Modeling Changes

Simplified Pad Modeling

Until now, you could only use a CPF macro model to model the IO pads. This method can be tedious and error-prone.

The define_pad_cell command allows you to create a model for simple pad cells. This command allows you to identify

- Pins that will be directly or indirectly connected to the board
- Pins which connect to the core and have internal isolation logic and their corresponding isolation control signal

- Groups of pins that belong to the same power domain
- Analog signal pins that connect to other analog pins

This model is not sufficient when your pad cell contains any of the following:

- Internal power switch and/ or state retention
- Complex internal isolation control
- Internal feed- through nets for data signals

In this case, you need to use a CPF macro model for the pad cell.

Simplified Pad Instantiation

The create_pad_rule command allows you to specify the mapping between a top-level domain and

- A pin group in the pad cell definition, or
- A power domain in the macro model definition of the pad cell

Improved Capability to Model an On-Chip Voltage Regulator

To model a voltage regulator the following features were added:

- Introduced the concept of a *power source domain*, that is a power domain that contains the output supply of the voltage regulator.
- Added the `-power_source` option to the create_power_domain to identify a power source domain.
- Added the set_power_source_reference_pin command to identify an input pin of a macro model as the voltage reference of the voltage regulator.
- Allowed specification of voltage ranges in the create_nominal_condition command.

Simulator-Related Changes

Added Capability to Specify Simulator Action when Power Domain is Switched off or Restored

To allow users to specify the immediate action to be taken when the power is switched off in power domains or restored to power domains, the set_sim_control command was added.

For example, initial statements are non-synthesizable code used in simulation to create proper startup conditions at time zero of the simulation. Previous CPF versions had no provision to specify the action to be taken when the power is restored to a switchable power domain. This time is very similar to initialization of the simulation at time zero.

Added Capability to Specify a CPF Model for a Testbench

To support this feature, the `-testbench` option was added to the set_design command.

Added Control over Signal Values in a Power Off Domain

To control the signal values of elements in a power domain when this domain is being switched off, the `-power_down_states` option was added to the create_power_domain command.

Other Improvements

Added Capability to Specify Inverted Polarity for Equivalent Control Pins

By allowing specification of equivalent pins with inverted polarity in the set_equivalent_control_pins command, insertion of special low power logic during synthesis can sometimes be simplified by using a control signal with the opposite polarity.

For example, with previous versions of CPF, when a limited number of isolation cells is available in the library, a combination of some conditions in the isolation rules might require the addition of an inverter to the control enable line. By using a control signal with the opposite polarity, the inverter is no longer needed.

Added Capability to Specify Libraries Specific to Power Analysis

To support this feature, the `-power_library_set` option was added to the create_operating_corner and create_nominal_condition commands.

Made Instance and Boundary Port Specification in Domain Specification More Flexible

To support this feature, the `-exclude_instances` and `-exclude_ports` options were added to the create_power_domain command.

Improved Support for Clamp Diodes

Previous versions of CPF only supported power clamp cells with one data pin

To improve the support of clamp diodes, the define_power_clamp_cell was superseded by the new define_power_clamp_pins command.

This new command supports

- Power, ground, and power and ground clamp cells with one or more data pins
- Complex cells that can have input pins with built-in clamp diodes

Added Support for Analog Ports in Macro Models

The set_analog_ports command allows you to specify a list of analog signal pins that must also be connected to pins declared as analog in other macro models or in pad cells.

Added Support for Deep Nwell and Deep Pwell

Added options `-deep_nwell` and `-deep_pwell` options to specify bias nets in the create_power_domain command.

Added options `-deep_nwell_voltage` and `-deep_pwell_voltage` options to the create_nominal_condition command.

Added Additional Operators for Boolean Expressions

The following grouping, bit-wise and logical operators have been added to the allowed operators in CPF Boolean expressions: ~ ^ && ||

Added Capability to Specify a Global Connection using Ports or Liberty PG Type

The create_global_connection command allows specification of a global connection between a net and either

- the top-level module’s ports, or
- an instance’s cell pins by matching the `pg_type` in the cell pins’ corresponding Liberty definitions.

Expanded Definition of Library Set for Operating Corners

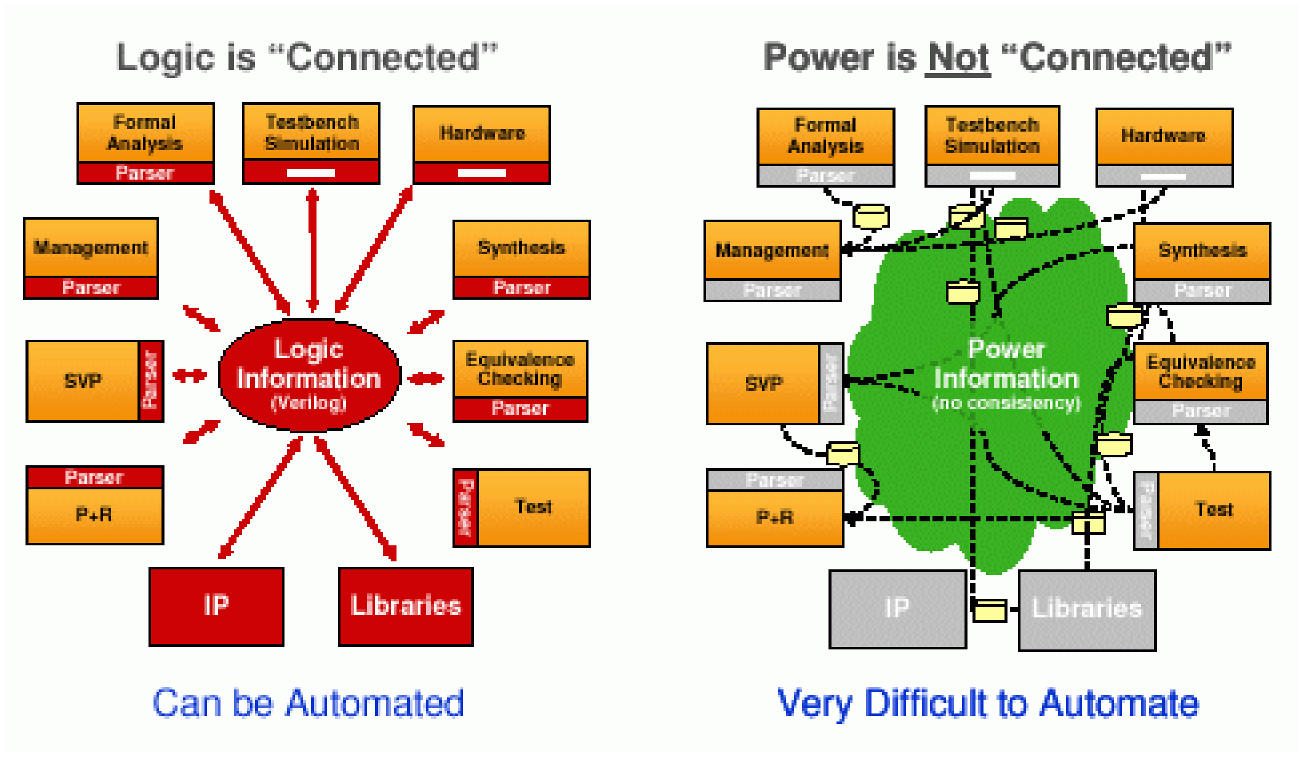
The create_operating_corner command now facilitates MMMC analysis and optimization by allowing the specification of more than one library set in the `-library_set` option.

1 Introducing the Common Power Format

Introduction

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 Power-Aware 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:

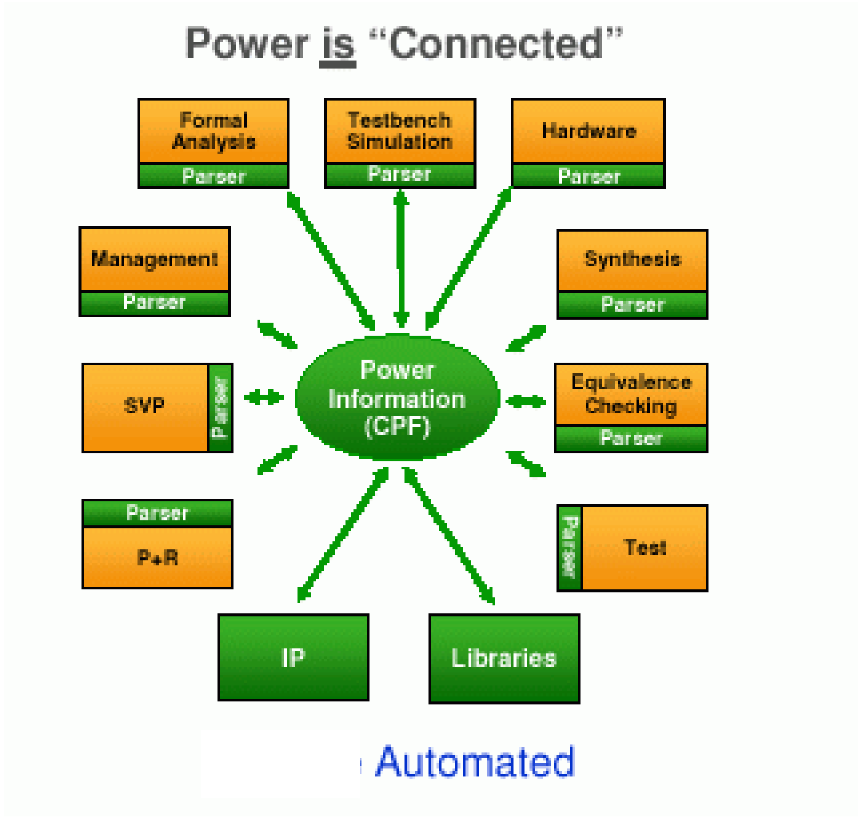
■ RTL/gate simulation	■ Physical synthesis / placement
■ Hardware simulation acceleration	■ Clock tree synthesis
■ Hardware emulation	■ Power grid design
■ Formal analysis	■ Power integrity analysis
■ Design analysis & rule checking	■ Design for Test
■ Formal verification	■ Automatic test pattern generation

■ 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 Benefit of the Common Power Format on the Design Flow



CPF Overview

CPF is a Tcl-based format Tcl **(http://en.wikipedia.org/wiki/Tcl)**. All of the power of Tcl may be used to express the intent, but a legal CPF file will resolve to just the commands enumerated in this specification. Commands that are specific to tools should be segregated into other files. Tcl libraries may be used to provide simplicity or standardization of expression, but these libraries must be provided to complete the design specification.

CPF commands are independent of any specific version of Tcl. If a CPF power description uses features unique to a specific version of Tcl, then it is the user's responsibility to ensure all tools in the tool chain support that version.

The CPF file specifies the power characteristics of a design, describing the design and its intended use such that tools may determine what additional elements must be added to support the correct operation of the design in a variety of power conditions. The CPF files for a design complement the RTL and/or netlist description of the design, and can be used throughout the design's creation, implementation, and verification flow.

This remainder of this document is organized as follows:

- Terminology defines many of the terms used throughout this document.
- CPF Syntax describes the syntax used to form a CPF command.
- CPF Example presents an example design and its CPF file.
- Power Domains discusses low power and how to model in CPF.
- Power Modes discusses power modes and power mode control groups.
- Precedence and Semantics of CPF Rules discusses specification of rules.
- CPF Semantics discusses general semantics for using CPF commands.
- General CPF Commands describes each CPF command in detail.
- Library Cell-Related CPF Commands describes commands that relate to library files.
- Quick Reference provides a list of all CPF commands and their arguments.
- Chapter by Chapter Summary of Changes provides a detailed list of section and command changes between the CPF1.1 and CPF 2.0 specifications.

2 Terminology

Hierarchy

Scope

Scope is defined as that portion of the design that is visible and may be affected by a CPF command.

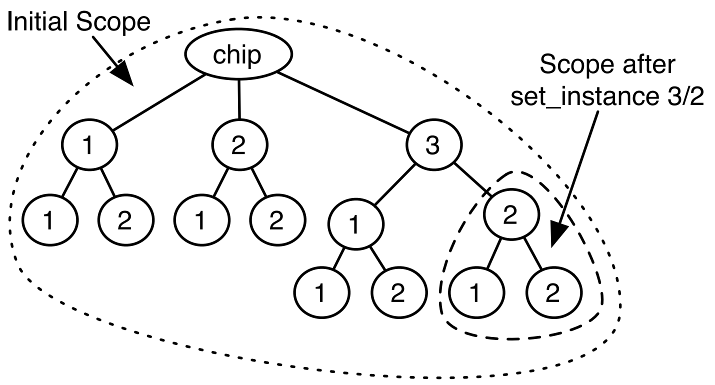
A hierarchical design can be represented as an inverted tree of instances, and the scope is initially set to the root of the tree. From the root, the entire design is visible, or “within scope”.

The set_instance command can be used to change scope. The new scope will be the instance specified in the `set_instance` and all instances below it in the design hierarchy.

For example, each circled portion of Figure 2-1 represents a scope.

Initially, the scope is the entire hierarchy from the top node down, and all elements are visible. After the `set_instance` command, the scope is reduced to a sub-portion of the design hierarchy and only the instances from 3/2 down are in scope

Figure 2-1 Scope



Top Level

Top level generally is a relative, scope-sensitive term referring to the topmost level of the hierarchy of a scope.

Top level does not usually refer to the topmost design of the chip or entire design hierarchy unless explicitly stated.

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 via their names.

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 one or more instance pins and/or ports.

A *net segment* is a connection between one driver and one load within the same module.

Pad

An instance of an I/O cell, also known 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 a CPF 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

For two power domains `X` and `Y`, where the primary power supply of `X` provides power to `Y` through a power switch network, the following relationships are defined:

- `X` is a ***base*** of `Y`
- `Y` is ***derived*** from `X`

Note: A power domain may have multiple bases.

Note: A power domain `X` may be both a base power domain for `Y` and be a derived power domain from another power domain `W`.

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.

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.

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.

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.

Conversely, 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, global 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.

Power Mode

A static state of a design in which each power domain operates on 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 only to this group. A power mode control group can contain other power mode control 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 Domains

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.

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. The cell function does not change when the primary supply is switched off as long as the secondary supply is still on.

A special case of a global cell.

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 a 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.

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, dual-rail isolation cells, etc.

Examples on page 259 illustrate CPF code to define global cells.

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

CPF Syntax

General CPF

CPF is embedded in Tcl. As such, CPF follows the syntax rules of Tcl (http://en.wikipedia.org/wiki/Tcl). CPF adds a number of commands to express the power intent associated with the RTL design.

A CPF command is composed of a command name followed by zero or more words that form arguments to the command. An argument consists of a value, a keyword, or a keyword-value pair.

- A value is a word representing a
  - string
  - integer
  - float
  - object(s) reference
  - a list of values
- A keyword identifies an option, and always contains a leading minus character (-). A keyword that is not immediately followed by a value is a simple flag.

Examples

```
set_design top
create_nominal_condition -name low -voltage 1.0
create_level_shifter_rule -name lsr1 -to {PD1 PD3}
```

Objects

The CPF file contains two categories of objects:

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

Object Names

CPF object names can contain

- any sequence of letters (of the alphabet),
- digits, the underscore (_),
- in some cases, the period (.), which is used only to express a nominal condition and operating corner name.

The name of a CPF object

- must be unique among CPF objects of the same scope
- may be the same as the name of a Design object.

Creating CPF Objects

Some CPF commands cause CPF objects to be created. A CPF command that creates a CPF object has the keyword *-name* followed by a string that is to be used as the object name. For example:

```
create_power_domain -name PD1 ...
```

In this example, a new CPF object, a power domain, is created with the name `PD1`.

Object References

Within a CPF file, commands may reference design objects and CPF objects. An object reference is always relative to the current scope.

The reference contains the object name, optionally preceded by a series of instance names separated by the hierarchical separator to describe the hierarchical path name of the object.

Some CPF commands allow the use of wildcards to reference multiple objects at a time.

Referencing Objects on page 107 details the semantics of specifying an object reference.

Compound Object References

Two CPF object names may be joined with the at character (@) to form a compound object reference. For example:

```
create_power_mode -name PM4 -domain_conditions {PD1@low}
```

Escape Character

The Tcl escape character is the blackslash character (\). It is used to escape special characters that have special meaning to the Tcl interpreter, such as square brackets ([) and (]).

The escape character is not required when the special character is contained within curly braces.

Wildcards

Wildcards used in an object reference specify multiple objects of the same type required by an argument. Wildcards are expanded only on objects within the current scope.

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

Example

Assume instances `A1`, `A2`, and `B1` exist in the current scope. Each is an instance of a module that contains two input pins `i1` and `i2`, and two output pins `o1` and `o2`. The hierarchical separator is the default (.).

- `A*` refers to `A1` and `A2`
- `*1` refers to `A1` and `B1`
- `*.i1` refers to three pins: `A1.i1`, `A2.i1` and `B1.i1`
- `A1.i*` refers to two pins: `A1.i1` and `A1.i2`
- `A1.o*` refers to two pins: `A1.o1` and `A1.o2`

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.

Hierarchy Separator

The supported hierarchy separators are

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

The hierarchical separator 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 separator character loses its meaning as a hierarchy separator.

The semantics of building an object reference using the hierarchical separator are specified in Referencing Objects on page 107.

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

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

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

Bus Delimiters

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

Because the square brackets ([ ]) are reserved characters in Tcl syntax, you must enclose the bus bit name in braces:

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

The square brackets only have this special meaning in object names. When the scalar object name has square brackets as part of the name, then each bracket character needs to be escaped and the name must be put within the curly braces, such as:

```
{a\[0\])
```

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]
```

A bus pin or bus port referenced without a range specification refers to all bits of the pin or port.

Individual Register Names

A flip-flop or latch instance name is based on

- A base name (the corresponding register name in RTL)
- (optional) A suffix appended to the base name

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 netlist.

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_analog_ports 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, and syntax is limited to the tokens shown in the following table:

Operator	Description
()	Parenthesis
~ & | ^	Bit-wise negation, AND, OR, XOR
! && ||	Logical negation, AND, 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

Important

- ❑ You can use parentheses () to change the operator precedence.
- ❑ 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`.

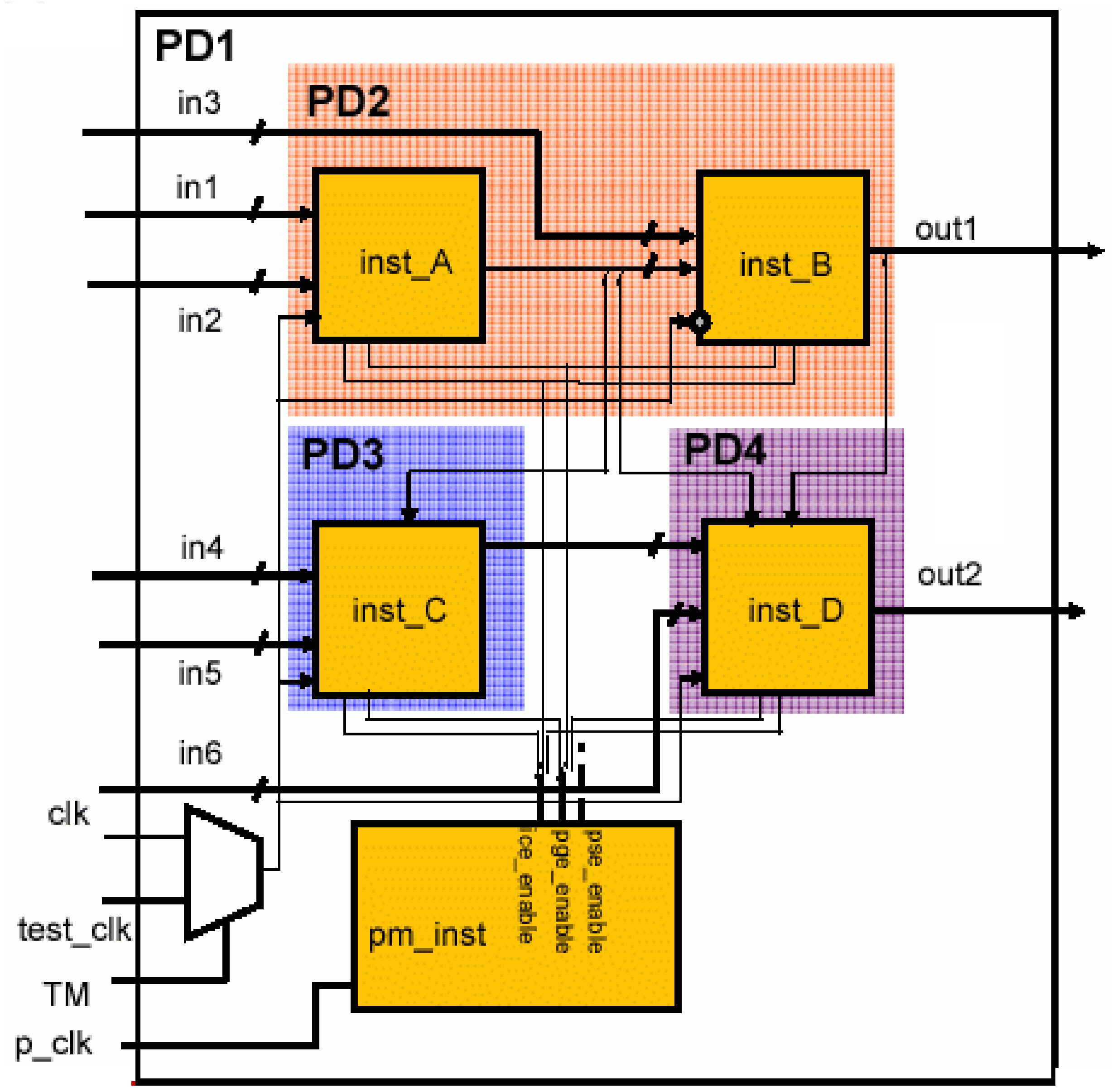
All voltage values must be specified in volts (V).

4

CPF Example

Consider the example design shown in Figure 4-1 on page 54.

Figure 4-1 Example Design for CPF



The design has four domains:

- The top-level of the design and hierarchical instance `pm_inst` belong to the default domain `PD1`
- Hierarchical instance `inst_A` belongs to the power domain `PD2`
- Hierarchical instance `inst_B` is an instantiation of an IP block. 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 4-1 on page 55 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 4-1 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 the pins `pse_enable`, `ice_enable` and `pge_enable`.

Table 4-2 Expressions Controlling the Power Domains

Power Domain	**Control Expressions**		
	power switch rule	isolation rule	state retention rule
PD1	no control signal	no control signal	no control signal
PD2	pm_inst.pse_enable[0]	pm_inst.ice_enable[0]	pm_inst.pge_enable[0]
PD3	pm_inst.pse_enable[1]	pm_inst.ice_enable[1]	pm_inst.pge_enable[1]
PD4	pm_inst.pse_enable[2]	pm_inst.ice_enable[2]	pm_inst.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 \
-default_restore_edge {!pm_inst.pge_enable[0]}
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
# 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 PD3 \
-restore_edge {!pm_inst.pge_enable[1]}
create_state_retention_rule -name sr2 -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
```

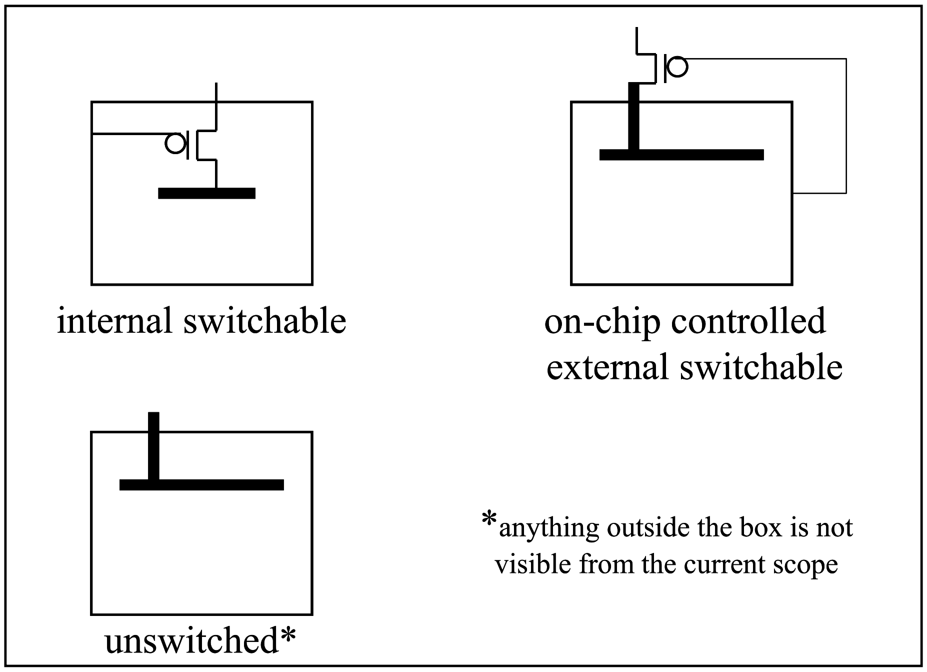
5

Power Domains

Power Domain Categories

CPF considers three categories of power domains as shown in Figure 5-1.

Figure 5-1 Categories of Power Domains


internal switchable
on-chip controlled external switchable
unswitched*
*anything outside the box is not visible from the current scope

As part of capturing design intent, the create_power_domain command may specify a shutoff condition that is independent of any specific implementation. The power and ground network must later implement that design intent (see create_power_nets and create_ground_nets). Verification tools should be able to check if the implementation matches the design intent.

unswitched domain

An unswitched domain is a domain that can not be powered down by controls or elements within the scope in which it was created.

Although an unswtiched power domain is not shut off in the scope where it is created, it may have 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.

To model an unswitched domain

- omit the `-shutoff_condition` option of the create_power_domain command.

internal switchable **domain**

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

To model an internal switchable domain

- Use the `-shutoff_condition` option of the create_power_domain command.
- Use the `-base_domains` option of the `create_power_domain` command to specify the domains from which the switchable domain gets its power supply.
- (optional) Define the on-chip power or ground switch using a power switch rule associated with the internal switchable domain for physical implementation.

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

- Use the `-shutoff_condition` option and the `-external_controlled_shutoff` options 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 must check the consistency between the shutoff condition of the power domain and the external shutoff condition of the primary power and ground nets.

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

If the external shutoff condition is defined for:

- only the primary power net, it must be identical to the domain shutoff condition
- only the primary ground net, it must be identical to the domain shutoff condition
- both the primary power and ground nets, their Boolean OR must be equal to the domain shutoff condition

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:

- `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

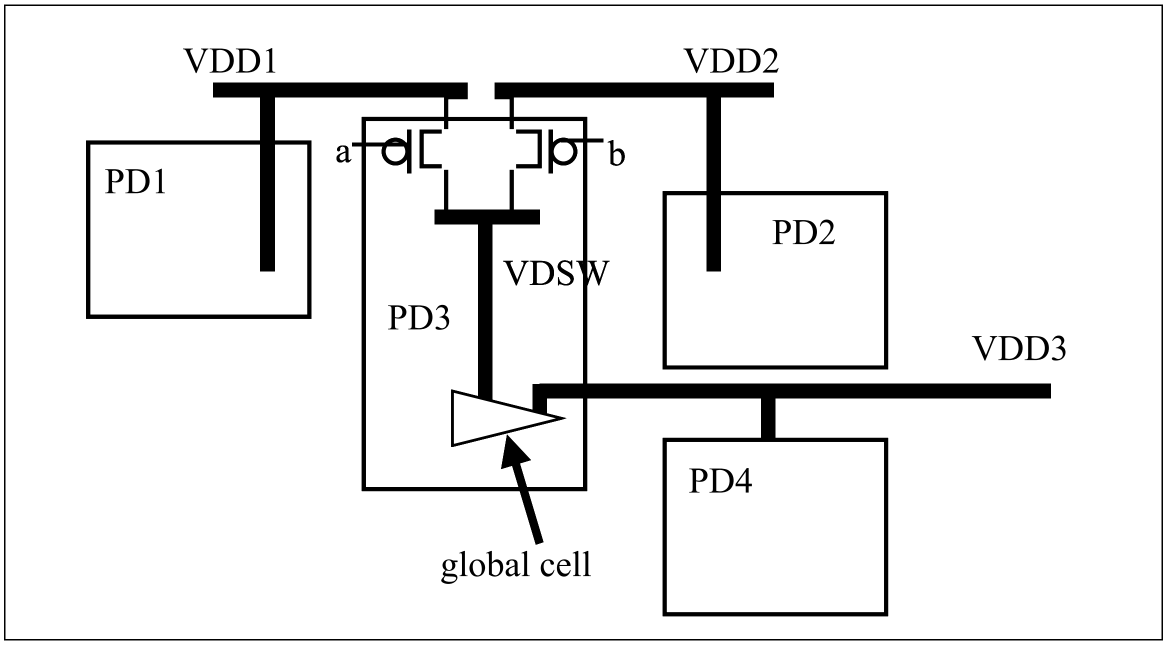
A power domain `X` is a ***base*** power domain of power domain `Y` if the primary power and ground nets of power domain `X` provide the power supply to domain `Y` through some power switch network.

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 5-2 has two base power domains, `PD1` and `PD2`.

Figure 5-2 Sample Design



For an internal switchable domain, an error should be given if the base domain 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 5-2 on page 63, 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`.

Primary and Secondary Power Domains

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.

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 global cell 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 5-2, `PD4` is not a base power domain for `PD3` even though `PD4` is the secondary domain for the global cell 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 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

It is an error in the following cases where the secondary domain of the isolation logic cannot be determined

- 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 for the state retention instance
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 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 Domain of Global Cells

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

During simulation, an instantiation of a global cell maintains its normal functionality as long as its secondary domain is on. If its secondary domain is shut off, the simulator must corrupt the output of the global cell instance

Note: In CPF version 1.0 and 1.1, the buffer and inverter type of global cell is called an always-on cell.

Input and Output Domains of Level Shifters

The definitions of primary and secondary power domains do not apply to level shifter instances. For level shifters, the 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 level case, the input power domain specification shall 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

Hierarchical Flow

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

Power Domain Mapping Concepts

In a hierarchical design flow, a block can be implemented with a complex power structure. The power and ground pins of the block must be connected to the power and ground nets in the top-level design that instantiates the block. In some cases, designers might want to merge a power domain of a block into a power domain that exists in the top-level domain where the block is instantiated.

Power domain mapping is the operation of merging a power domain of a block into a power domain that exists in the top-level design where 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 top-level domain
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 together to the same primary power and ground nets.

Note: It is an error if the default block-level domain is mapped into a top-level domain that is different from the domain to which the block instance is assigned at the top level. 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.

Handling Power Domain Mapping

If the top-level domain is unswitched, all rules defined for the block-level domain need to be re-evaluated to see if they should be ignored.

However, if the top-level domain is switchable, the verification tools must 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 two 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 195.

  b. The two expressions become identical when 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 specified, the number of base domains of the block level must equal the number of base domains of the top-level domain. In addition, each base domain of the block-level must map into a unique base domain of the top-level domain.

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 is an internal switchable domain.
- Both the top-level and block-level domain have exactly one base domain.

- The base domain of the block-level domain is mapped into the base domain of the top-level domain.
- The domain shutoff conditions of the top-level and block-level domains are structurally equivalent.

Handling Power Modes in Power 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:

  - It is an error if a power mode at the top level does not match any corresponding power modes at the block level. Such a scenario should be avoided since it indicates an IP may be used in a wrong way.
  - It is a warning if a power mode at the block level does not match any corresponding power modes at the top level to 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: Note for each scope, an implicit power mode definition exists where all power domains at that scope are operating in the `off` state. See Power Mode for details.

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

  In this case, the 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 domains directly in the domain condition specification, or refer to a block-level mode (that involves these domains) using the power mode control group definitions. See Power Mode Control Groups for details.

CPF Modeling for Hierarchical Design

In a hierarchical design flow, a block can describe its power intent in a separate CPF file. The block-level CPF is referred to as the ***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 file with the block definition, then use the set_instance command with the
  - `-model` option to reference a CPF macro model for a macro cell
  - `-design` option to reference a CPF power design model for a module (see Example 5-1 on page 73)

  Note: A `set_instance` command specified with an instance name and a `-design` or `-model` option will not cause a scope change.

- Use the `set_instance` command without the above options followed 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 set_instance on page 202.

  In this case (see Example 5-2 on page 73) 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 applied to the specified instance.

  c. The scope will be restored 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.

Both macro model and power design definitions are scope sensitive.

- It is legal to have two macro models or two power designs created with the same name *in different scopes*.

- If there are more than one macro model or more than one power design declarations with the same name *in the same scope*, only the first model will be used. Subsequent models will be ignored, and a warning shall be issued. See Example 5-3 on page 74, Example 5-4 on page 74, and Example 5-8 on page 75.

This behavior is changed from CPF 1.1, in which 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.

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.)

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.

Note: 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 named 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 are bound to the first model of foo, while I2 is 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 is linked to the first CPF macro model for foo. The second macro model is ignored and a warning is 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 that 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 # appended 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 updated second definition of model foo is appended to the content of the original 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 there is no CPF model named `foo` 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 `I2`'s instantiation.

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 additional 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
...
end_design foo
set_design nested_design
...
set_design 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 there are two power designs with the same name `foo` 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
...
end_design foo
set_instance N1 -domain_mapping ...
set_design nested_design
...
set_design foo
...
end_design foo
...
end_design nested_design
set_instance I1 -design foo -domain_mapping ...
...
end_design top
```

This example is almost the same as Example 5-8 except that a `set_instance` statement has been added before `set_design nested_design`. Because `set_instance` causes a scope change, the two definitions `foo` are different power designs. The first definition belongs to the top scope, and the second definition belongs to the scope of `N1`.

Instance `I1` refers to the power design `foo` belonging to the top scope.

6

Power Modes

Nominal Condition

A nominal condition is a CPF object that defines the power supply characteristics for one or more power domains.

A nominal condition can be as simple as specifying the power supply voltage or as complex as specifying supply power and ground voltages and PMOS and NMOS bias voltages. Each nominal condition must be specified as one of the following states:

- `off`
- `on` (or operational)
- `standby`

Off State

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

Example

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

In this example the nominal condition `power_down` is defined. The `power_down` nominal condition has a state of `off` and a power supply voltage of 0.0V.

On State

The logic of a power domain in an `on` state is fully operational.

A nominal condition with an `on` state can have a different set of supply voltages. The different supply voltages will determine the performance characteristics of the devices in a power domain. For example, by applying a forward body bias, a CMOS device will switch faster than when normal body bias is applied.

Example

```
create_nominal_condition -name slow -voltage .8 -state on
create_nominal_condition -name normal -voltage 1.0 -state on
create_nominal_condition -name fast -voltage 1.2 -state on
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

The logic of a power domain in a `standby` state maintains its logical 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 or source biasing. (see create_nominal_condition)

Examples

```
    1. create_nominal_condition -name sleep1 -voltage .5 -state standby
    2. create_nominal_condition -name sleep2 -voltage 1.0 -state standby \
        -pmos_bias_voltage 1.2
```

In example `1` the nominal condition `sleep1` is created as a `standby` state using source biasing.

In example `2` the nominal condition `sleep2` is created as a `standby` state using reverse body biasing.

Generic Mode

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

The power mode can be considered as a special case of mode, where the state of a design is determined exclusively by the nominal conditions of all power domains of the current scope.

Unlike a power mode that 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 a state 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 }
```

Modes are useful for describing high-level functional states of the design. Due to shrinking geometries and exponential increase in transistor counts on modern day chips, most designers build power saving features and techniques into their functional designs. The example below shows how CPF modes can be used to describe this.

```
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.

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

Power Mode

Each power mode represents a legal configuration of all power domains in a scope operating at specific nominal conditions.

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

Example

```
create_power_mode -name off -domain_conditions {PD1@sleep PD2@sleep}
create_power_mode -name drowsy -domain_conditions {PD1@sleep PD2@normal}
create_power_mode -name on -domain_conditions \
        {PD1@normal PD2@normal PD3@normal}
```

For example, a `drowsy` mode is created by assigning power domain `PD1` to nominal condition `sleep` and power domain `PD2` to nominal condition `normal`.

Note: A power mode requires the specification of the state of every power domain in the current scope. An unreferenced power domain is considered `off`. For example, `PD3` is `off` in `drowsy` mode, above. This requirement is one of the main differences between a power mode and a generic mode.

In addition to power domains operating at specific nominal conditions, a power mode may also contain any number of power mode control groups operating at specific power modes. Power mode control groups are used in a hierarchical flow where a piece of reusable IP has defined power modes in the CPF file for that IP.

An Illegal Domain Configuration

The assert_illegal_domain_configurations command is used to specify that a particular configuration of domain conditions and/or power mode control group conditions is illegal. For example, to safeguard the case where an unswitched power domain may not turn off in relation to other power domains (to prevent unisolated internal gates or a high current situation for example), the assert_illegal_domain_configurations command can be used to declare explicitly a power mode or a configuration of domain conditions to be 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.

Example 6-1 Illegal Domain Configuration

In the following example, the command declares that it is illegal to have power domains `PD1` and `PD2` at the nominal condition `nc1` at the same time.

```
assert_illegal_domain_configurations -name fooAssert \
    -domain_conditions {PD1@nc1 PD2@nc1}
```

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.

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 (see Referencing Objects) from the current scope to the targeted scope.

The default power mode control group contains all those power domains defined in the scope that are not declared as members of an explicitly defined power mode control group, and if each power domain of a scope belongs to an explicitly defined power mode control group, the default power mode control group has no members.

Explicitly Created Power Mode Control Groups

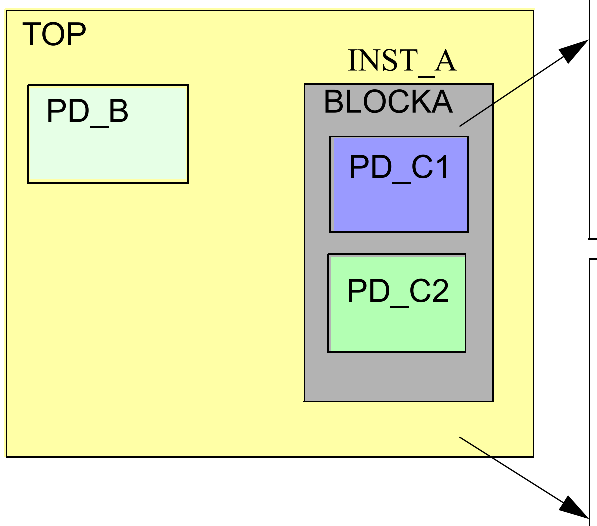
The set_power_mode_control_group and end_power_mode_control_group commands define the start and end of an explicit power mode control group definition.

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

At a given scope, one or more power mode control groups may be created. A power mode control group can be referenced by top-level CPF commands using its hierarchical name.

For backward compatibility, a power domain that is not part of any power mode definition within a power mode control group is assumed to be powered down. (In CPF 1.0, a power domain that is missing from a power mode definition is assumed to be powered down.)

Example 6-2 Use of 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
```

In Example 6-2 on page 85, `BLOCKA` 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`, a default power mode control group is created. 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 control mode 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 referred to in this top-level mode, it is assumed to be in the default mode of the power mode control group, which is mode `M1`.

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.

Example 6-3 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 6-4 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 instantiation of `BLOCKQ` from the previous example illustrates references to the power mode control groups within `BLOCKQ`.

Example 6-5 User-Created Power Control Groups with Domains in Multiple Groups

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

Mode Transition

A mode transition describes a transition from one power mode to a different power mode. This transition may be characterized by specifying the number of cycles or range of cycles required to make the transition. Alternatively, a latency can be specified as either a specific number or range of numbers that describe the transition from one power mode to the next. Mode transitions can also have a start condition and end condition that describe the condition to trigger the transition and the condition to flag the end of the transition. These conditions are Boolean expressions, built from the ports or pins in the design.

7

Precedence and Semantics of CPF 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.
    - State retention rules specify the targeted design objects with the `-instances` option.
- **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 those net segments 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.

Precedence of Rules in the Same Scope

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 general rules of precedence apply:

1. Generic rules specified with both the `-from` and `-to` options have a higher priority than those generic rules specified with only `-from` or `-to` option.
2. Specific rules have a higher priority than generic rules.
3. When multiple CPF rules with the same priority are applied to the same design objects, precedence is determined as described in Information Precedence.

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 should be followed in hierarchical flow:

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 rules are never propagated to the top level.

Rules Semantics

- Level Shifter Rules
- Isolation Rules
- State Retention Rules

Level Shifter Rules

A level shifter rule targets one or more nets that require level-shifter logic. Level-shifter logic can be required because the leaf driver and leaf loads of the net have different supply voltages, or because it is part of the design intent.

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

Note: It is an error to apply a level-shifter rule to a bidirectional port.

When Can Implementation Tools Ignore a Level Shifter Rule

Level shifter rules can be ignored in the following cases (unless the `-force` option is specified):

- 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` – instantiates the level shifters 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.

- `to` – instantiates 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` – instantiates the level shifters in the logic hierarchy as determined below:
  - ❑ 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
- `any` – works only with the `-within_hierarchy` option to indicate that the level shifter may be instantiated 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 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 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 targets one or more nets that 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.

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]
```

In the 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

Isolation rules can be ignored in the following cases (unless the `-force` option is specified):

- 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.

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.

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` – instantiates 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.
- `to` – instantiates 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` – instantiates the isolation logic in the logic hierarchy as determined below:
  - 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
- `any` – works only with the `-within_hierarchy` option to indicate that the isolation logic may be instantiated in any power domain to which the specified logic hierarchy belongs.

The location for the inserted isolation instances 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 on 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.

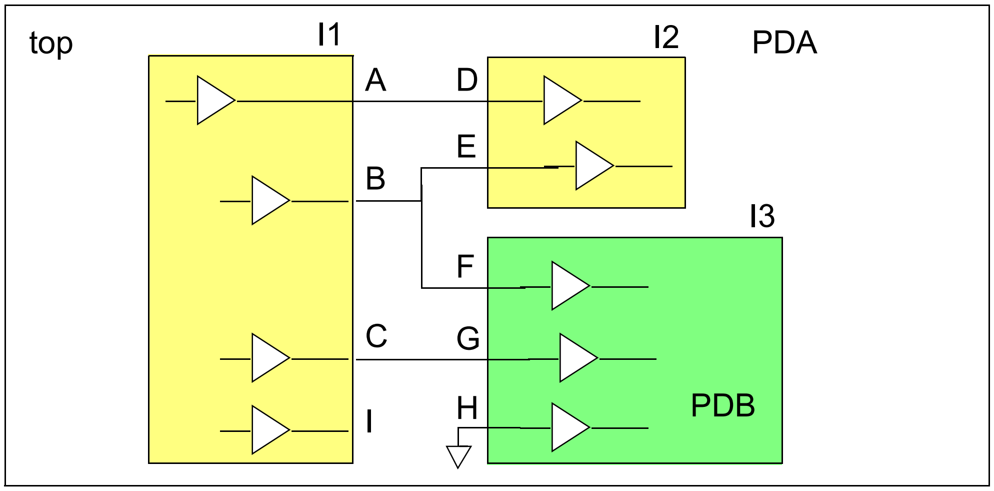
The `off` value for `-valid_location` is not commonly used and will be deprecated in a future release. All isolation cells can be described with the above four different valid locations.

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 7-1, instances `I1` and `I2` belongs to power domain `PDA,` and instance `I3` belongs to power domain `PDB`.

Figure 7-1 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 7-1 because it is a floating net.

- Rule `iso2` can be ignored for the net connected to pin `H` in Figure 7-1 because it is driven by a tie-low signal.
- Net segments `AD` and `BE` in Figure 7-1 should not be considered for any rule because the leaf-level driver and loads are in the same power domain.

State Retention Rules

A state retention rule targets 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.

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 condition is

- defined, this condition can be used by the incomplete state retention rule to make the rule complete.
- undefined, the incomplete state retention rule remains incomplete.

  Note: An incomplete rule will be ignored by the implementation tools.

When Can Implementation Tools Ignore a State Retention Rule

A tool does not need to implement a state retention rule if:

- The save or restore condition for the rule cannot be determined.
- The rule is specified for a domain that is never off in all the mode definitions, except for all domains off.
- There is a state retention cell that implements this rule.
- The state retention rule is specified with virtual ports at the block level, but the virtual ports are mapped to constants (or constant signals) at the top level.

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.

8 CPF Semantics

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
	end_design
	update_design
	get_parameter
	set_instance
macro support	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
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

Category	CPF Command
	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
	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

Category	CPF Command
	define_related_power_pins
	define_state_retention_cell

Tip

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

Information Precedence

- Multiple update commands may be used to add implementation details for CPF objects as long as each command specifies unique information. If the same information is specified, the information specified in the last command takes precedence, unless specified otherwise.

  For example, the result of the following 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.

  ```
  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
  ```

- If information defined in the CPF file conflicts with information in the referenced library, the information in the CPF file takes precedence and a warning shall be issued.

Important

Currently, if you define a CPF object in a specific scope multiple times with the same name, the last definition takes precedence unless specified otherwise. Use of this feature is not recommended, and future releases of CPF may not allow it.

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
```

```
set_design Top
# The following command sets the hierarchy separator 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
set_design A
...
# the hierarchy separator is still . according to the inheritance rule.
...
end_design
...
end_design
```

Referencing Objects

Objects are referenced by a string that uniquely identifies the object.

An object reference specifies the object name, optionally preceded by path. A path is a series of instance names separated by the hierarchal separator. The path is not required if the object is at the same level of hierarchy. If the object is lower in the hierarchy, the path contains the instance names through the hierarchy from the current scope to the object being referenced.

Only objects within the current scope or lower may be referenced, and the current scope is always considered to be the top.

A hierarchical separator at the beginning of a path denotes the top of the current scope. Since an object reference is always from the top of the scope, the separator may be omitted. For example, `a/b` and `/a/b` refer to the same object.

Example

```
1. set_instance instA
2. set_design instA
3. create_power_domain PD1
4. end_design
5. create_isolation_rule -from instA.PD1
```

Line number 5 refers to object `PD1` in `instA`.

Referencing Design Objects

When you reference any design object other than a pin, port, or net, that object must exist in the design.

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 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 using the `-ports`, `-input_ports`, `-output_ports`, or `-inout_ports` option of the set_design command.

Example

```
-isolation_condition {pm_inst.ice_enable[0]} -isolation_output high
```

In this example, bit 0 of pin `ice_enable` of the instance `pm_inst` is referenced, and `pm_inst` must be visible in the current scope.

Referencing a Power Supply Net

When you reference a power supply net,

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

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

- A CPF object can be referenced only after it is created
- To reference a CPF object created at the current scope, use the name without a hierarchical path.
- To reference a CPF object created *below* the current scope, use the name of the object preceded by a hierarchical path.
- 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. Now 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).

9

General CPF Commands

- update_design on page 227
- update_isolation_rules on page 228
- update_level_shifter_rules on page 233
- update_nominal_condition on page 237
- update_power_domain on page 238
- update_power_mode on page 244
- update_power_switch_rule on page 248
- update_state_retention_rules on page 251

assert_illegal_domain_configurations

```
assert_illegal_domain_configurations
     -name string
     { -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/or power mode control group conditions is illegal.

The assertion is only checked against conditions explicitly specified. In other words, no assumptions are made about unspecified domains.

A validation tool will verify that the design never enters a power mode that matches all of the domain and group conditions.

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

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

`-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.

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. **Note:** The specified string cannot contain wildcards. **Note:** The specified string cannot contain the hierarchy delimiter character.

Related Information

Power Mode on page 82

create_power_mode on page 158

Power Mode Control Groups on page 84

create_analysis_view

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

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 may contain no analysis views, but if any are specified, one and only one will be the default. If the user does 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.

`-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 or the hierarchy delimiter character.

`-user_attributes` *`string_list`*

Attaches a list of user-defined attributes to 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 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_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 36

create_operating_corner on page 145

create_power_mode on page 158

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.

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.

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.

`-exclude` *`assertion_list`*	
	Specifies a list of assertions to be excluded from the selected assertions (by -`assertions` or -`domains` option).
`-name` *`string`*	Specifies the name of the assertion control.
	Note: The specified string cannot contain wildcards.
	Note: The specified string cannot contain 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`

Related Information

create_power_domain on page 150

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.

Options and Arguments

`-average_ir_drop_limit` *`float`*	Specifies the maximum allowed average voltage change on a bias net due to resistive effects in volts (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.
`-peak_ir_drop_limit` *`float`*	Specifies the maximum allowed peak voltage change on a bias net due to resistive effects in volts (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 125

create_global_connection

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

Specifies how to connect a global net to the specified pins or ports. 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, that pin is ignored for connection while the remaining pins are connected to the specified global net.

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` *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 LEF pins 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 pin names.

Related Information

Wildcards on page 46

create_bias_net on page 123

create_ground_nets on page 127

create_power_domain on page 150

create_power_nets on page 161

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 volts (V) for any mode.

Default: `0`

`-external_shutoff_condition` *`expression`*

When a specified ground net is connected to an external source, you can use an 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.

`-peak_ir_drop_limit` *`float`*

Specifies the maximum allowed peak ground bounce on the specified grounds net due to resistive effects in volts (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}...
     [-exclude pin_list]
     [-isolation_target {from|to}]
     [-isolation_output { high | low  | 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 specifies which net segments must be isolated. You can

- Select all net segments driven by logic in the domains specified with the `-from` option *and* that are driving logic in other power domains
- Select all net segments 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 net segments:
  - If you combine `-to` and `-from` options, the rule should apply to those net segments 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 net segments 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 net segments 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 net segments 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

Important

If you use the `-pins` option to select the net segments to be isolated, you must always combine this option with the `-from`, `-to` or `-from` and `-to` options.

Options and Arguments

`-exclude` *`pin_list`* Excludes from isolation those *already selected* net segments that are connected to, driven by, or driving the specified pins.

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 95).

It is the responsibility of the implementation tool to perform optimization that avoids the redundant insertion of isolation logic.

`-from` *`power_domain_list`*

Specifies that the rule is to be applied to those net segments 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 net segments should be isolated. The condition can be a 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 net segments 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 net segments will be used as the isolation condition.

If the `-default_isolation_condition` option for the driving power domain of the selected net segments was not specified, the incomplete rule is treated as a design constraint.

`-isolation_control` *`list_of_additional_controls`*

Specifies a list of controls, where the format of each control is
{ *`control_type expr`* }

- *`control_type`* is a key word to indicate the type of isolation control Allowed types are
  - `sync_enable` – can only be used with isolation types `high` and `low`. The corresponding *`expr`* needs to be true before the isolation condition can be asserted and deasserted. But the expr can be corrupted when the isolation condition is asserted.
  - `set` – can only be used with `-isolation_output hold`. When the *`expr`* 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 *`expr`* 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, there must exist an isolation cell definition in which the `-aux_enables` option specifies only one pin, and that pin must relate to the switchable supply.

`-isolation_output {high|low|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`

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.

Note: The specified string cannot contain the hierarchy delimiter character.

`-no_condition`	Specifies that the isolation logic is automatically enabled when the power domains containing the drivers of the selected net segments 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 that the rule is to be applied to those net segments 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 is to be applied to those net segments 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.

Related Information

Isolation Cell on page 40

create_power_domain on page 150

update_isolation_rules on page 228

create_level_shifter_rule

```
create_level_shifter_rule
     -name string
     {-pins pin_list | -from power_domain_list | -to power_domain_list}...
     [-force]
     [-exclude pin_list]
     [-bypass_condition expression]
     [-output_domain power_domain] [-input_domain power_domain]
```

Defines a rule for adding level shifters.

A specific rule allows you to indicate on which net segments to insert level shifters. You can

- Select all net segments driven by logic in the domains specified with the `-from` option *and* that are driving logic in other power domains
- Select all net segments 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 net segments:
  - If you combine `-to` and `-from` options, the rule should apply to those net segments 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 net segments 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 net segments 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 net segments 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

Important

If you use the `-pins` option to select the net segments for level shifter insertion, you must always combine this option with the `-from`, `-to` or `-from` and `-to` options.

Tip

The `-input_domain` and `-output_domain` options are useful to connect 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 the voltage shifting functionality should be bypassed. 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 in `define_level_shifter_cell`.
`-exclude` *`pin_list`*	Excludes from level shifter insertion those *already selected* net segments that are connected to, driven by, or driving the specified pins. 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 shall be inserted, even in situations in which the rule would normally be ignored (see When Can Implementation Tools Ignore a Level Shifter Rule on page 92). Implementation tools should be able to perform optimization to avoid the redundant insertion of level shifters.
`-from` *`power_domain_list`*	
	Specifies that the rule is to be applied to those net segments 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 shall be ignored.

`-name` *`string`* Specifies the name of the level shifter rule.

Note: The specified string cannot contain wildcards.

Note: The specified string cannot contain the hierarchy delimiter character.

`-output_domain` *`power_domain`*

Specifies the output power domain of the level shifters inferred by this rule.

By default, the output 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.

Note: If an inferred level shifter drives logic that belongs to multiple power domains with different supplies, this option must be specified. Otherwise it is an error.

`-pins` *`pin_list`* Specifies that the rule is to be applied to those net segments that are connected to, driven by, or driving the specified pins.

You can specify ports and instance pins.

`-to` *`power_domain_list`*

Specifies that the rule is to be applied to those net segments 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.

Related Information

Level Shifter Cell on page 40

When Can Implementation Tools Ignore a Level Shifter Rule on page 92

create_power_domain on page 150

define_level_shifter_cell on page 266

identify_power_logic on page 182

update_level_shifter_rules on page 233

create_mode

```
create_mode
     -name string -condition expr
     [-probability float] [-illegal]
```

Defines a mode of the design. Modes can be non-mutually exclusive, so multiple modes may 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` *expr*	The condition when the design is in the specified mode, described by a Boolean or arithmetic expression of the following objects: ■ design pins and ports ■ domain conditions in the form of *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
  - M2, when
    - a&b is true, or

- PD1 is at nom1, PD2 is at nom2, and the design is in mode M1

```
create_mode -name M1 -condition { PD3@nom1 }
create_mode -name M2 -condition { (a&b) | ((PD1@nom1 & PD2@nom2) & @M1) }
```

Related Information

create_power_mode on page 158

create_mode_transition

```
create_mode_transition
     -name string
     -from mode -to 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.

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.

`-end_condition` *expression*

Specifies the condition that acknowledges that the design is in the mode specified by the `-to` option.

`-cycles [`*integer*`:]`*integer*

Specifies an integer of number of clock cycles needed to complete the 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.
`-from` *`mode`*	Specifies the mode from which to transition.
	The mode must have been previously defined with `create_mode` or `create_power_mode`.
`-illegal`	Identifies the mode transition as an illegal one.
`-latency [`*`float`*`:]`*`float`*	
	Specifies the time needed to complete the 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 mode transition.
	Note: The specified string cannot contain wildcards.
	Note: The specified string cannot contain the hierarchy delimiter character.
`-start_condition` *`expression`*	
	Specifies the condition that triggers the mode transition.
`-to` *`mode`*	Specifies the mode to which to transition.
	The mode must have been previously defined with `create_mode` or `create_power_mode`

Related Information

Power Mode on page 82

create_mode on page 138

create_power_mode on page 158

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:

$$min \le voltage < max$$

Note: A power domain is switched off if the nominal voltage value in option `-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 a 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 volts (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 {`*`voltage`*` | `*`voltage_list`*`}`

Specifies either a single voltage or a 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 volts (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` *`float`*	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 volts (V).

Related Information

Nominal Operating Condition on page 37

create_power_mode on page 158

update_nominal_condition on page 237

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 volts (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 `-power_library_set` 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.

Note: The specified string cannot contain 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 volts (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 volts (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 in `-library_set` 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 volts (V).

Example

The following illustrates the specification of multiple library sets in creating 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 ... -voltage ... -library_set {set1 set2}
```

Related Information

Operating Corner on page 37

define_library_set on page 172

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 must be an instantiation of a pad cell or a CPF macro model.

`-name` *`string`* Specifies the name of the pad rule.

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

`-mapping` *`mapping_list`*

Specifies the mapping of each pin group of the pad cell definition or a power domain in the 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.

`-of_bond_ports` *`port_list`*

Selects pad cell instances that are connected to the specified list of top-level bonding ports.

A selected instance must be an instantiation of a pad cell or 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_cell -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 275

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 {high|low|random|inverted} ]
     [-power_down_states {high|low|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.

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 or macro model, and one (and only one) power domain must be specified as the default power domain.

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 standby.

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 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 to the domain through some power switch network.

When all base power domains are switched off, the domain will be switched off irrespective of the shutoff conditions.

`-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 and outputs 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: Hierarchical instance pins will be ignored if specified.

`-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 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 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`*

Excludes 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`*

Excludes 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 a on-chip external switchable power domain. The presence of this option creates a requirement for validation of an external switch.

`-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.

Note: The specified string cannot contain the hierarchy delimiter character.

`-power_down_states {high|low|random|inverted}`

Specifies the value with which to overwrite the default corrupt semantics of elements 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 omitted, simulation tools should use the default power-down semantics for simulations.

`-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.

Important

This option can only be specified for a power domain within a macro model.

`-power_up_states {high | low | random | 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 this option is omitted, 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.

If this option is omitted, the power domain is considered to be unswitched.

If both `-shutoff_condition` and `-external_controlled_shutoff` are specified, the power domain is considered to be on-chip controlled external shutoff.

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 I1 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}}
  ```

- 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 a power source domain.

Related information

Power Domain on page 38

Referencing CPF Objects on page 110

create_global_connection on page 125

create_isolation_rule on page 129

create_nominal_condition on page 142

create_power_switch_rule on page 163

create_state_retention_rule on page 165

update_power_domain on page 238

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.

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.

`-name` *`string`*	Specifies the name of the mode. **Note:** The specified string cannot contain wildcards. **Note:** The specified string cannot contain the hierarchy delimiter character.

Examples

- In the following example, the nominal conditions that power domain `PD1` is associated with have an overlapping voltage range [`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 39

create_nominal_condition on page 142

create_power_domain on page 150

set_pad_ports on page 211

update_power_mode on page 244

Power Mode Control Groups on page 84

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 volts (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 always 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.

`-peak_ir_drop_limit` *`float`*

Specifies the maximum allowed peak IR drop on the specified power nets due to resistive effects in volts (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 41

create_power_domain on page 150

define_power_clamp_cell on page 278

update_power_domain on page 238

update_power_switch_rule on page 248

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 that the rule is to be applied 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`*

Excludes 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 hierarchical instance name in a gate-level netlist
- Register variable or hierarchical 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 hierarchical instance name in a gate-level netlist
- Register variable or hierarchical 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.

Note: The specified string cannot contain the hierarchy delimiter character.

`-required`

Indicates that the registers selected by this rule must be implemented using retention flops or latches.

If this option is not specified, then 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 regardless of whether or not `-required` is specified.

It is an error if a register is selected by an incomplete retention rule (see State Retention Rules on page 99) with `-required` specified, and

- there is no other complete retention rule to select this register, or
- there is no default condition specified for the domain that owns the register, i.e. `-default_save_edge` in `create_power_domain`

`-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.

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.

`-restore_precondition` is intended for RTL simulation only.

`-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.

Note: `-save_precondition` is intended for RTL simulation only.

`-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 types `flop` and `latch` can be converted.

Default: `flop`

`-use_secondary_for_putput`

When this option is specified, the output of the retention logic will keep the previous value when the primary domain is shutoff and 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 shutoff and the secondary domain is on. In other words, the output of the instance associated with the primary domain of the instance.

If both the primary and the secondary domain are off, then the output will be corrupted.

Related Information

State Retention Cell on page 41

State Retention Rules on page 99

create_power_domain on page 150

define_state_retention_cell on page 286

update_state_retention_rules on page 251

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 37

Library Set on page 37

end_design

```
end_design
    [power_design_name ]
```

This command is paired with a set_design or update_design command to group a set of CPF statements 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 `set_design`. or `update_design`. When the power design name is specified, it must match the power design name of the closest `set_design` or `update_design` command. Otherwise it is an error.

Related Information

CPF Modeling for Hierarchical Design on page 72

set_design on page 190

update_design on page 227

set_instance on page 202

end_macro_model

```
end_macro_model
    [macro_model_name]
```

This command is paired with a set_macro_model command to group a set of CPF statements defining 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 `set_macro_model`. **Note:** When the macro model name is specified, it must match the macro model name of the closest `set_macro_model` command. Otherwise it is an error.

Related Information

set_macro_model on page 208

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

set_pad_ports on page 211

Power Mode Control Groups on page 84

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}`

Indicates that only pins or ports with the specified direction will be returned by this command.

Applies only if `-object` specifies `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 `-direction` is not specified and `-object` is either `pin` or `port`, all matching pins or ports will be returned, irrespective of direction.

`-exact`	Indicates that only objects whose names exactly match the *`pattern`* value will be returned. No wildcard expansion is performed.
`-hierarchical`	By default, the search only applies to the specified scopes, not including its children. When this option is specified, the command will recursively search the specified scopes and all children of those scopes.
`-ignore_case`	Indicates that the search is case insensitive 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 `-object` specifies `inst` or 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` – return instance objects.
- `port` – return only port objects of the modules in the current scope. When this type of object is specified, the `-scope` and `-hierarchical` options are ignored, and tools can choose whether or not to report a warning in this situation.
- `pin` – return pin objects.
- `net` – return net objects.

Default: `inst`

`pattern` Specifies the search string. By default, it is a Tcl glob expression.

`-pattern_type` Indicates the type of name to be matched by the pattern string.

- `name` – allowed for any type of object specified by `-object`. Returns objects whose names match the *`pattern`*.
- `cell` – allowed only if `-object` specifies `inst` or is omitted. Returns objects that are instantiations of library cells whose names match the *`pattern`*.
- `module` – allowed only if `-object` specifies `inst` or is omitted. Returns objects that are instantiations of modules whose names match the *`pattern`*.

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 shall start.

When not specified, the search begins in the current scope.

Only the current scope and its children can be referenced.

A list of hierarchical instances can be specified, but no wildcard characters or regular expressions are allowed.

It is an error if the specified hierarchical instances cannot be found within the current scope.

Example

Given the example hierarchy, and assuming the current scope is at `top`:

```
 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 -type inst -hierarchical
   -> 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

Scope on page 33

Instance on page 34

Module on page 34

Net on page 34

Pin on page 35

Port on page 35

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 -input_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 190

set_instance on page 202

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 must already exist in the RTL or netlist.

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

Base and Derived Power Domains on page 36

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 directory of the top level CPF file being read and ".." refers to the parent directory of the top level CPF file being read.

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 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
- analog pins in a pad cell.

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 275

set_macro_model on page 208

end_macro_model on page 174

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.

Note: This command is only needed to specify RTL-to-netlist name conversion, and only applies if the design is RTL.

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:* `\[%d\]`

Related Information

Specifying the Representation of the Bits on page 49

Information Inheritance on page 106

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.

Options and Arguments

value	Specifies the version. Use a string. *Default*: `1.1`

set_design

```
set_design
     power_design_name [-modules module_name_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 that each boundary port domain assignment is to be treated 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_name_list*

This option specifies the names of the modules to which the power design applies.

If this option is omitted, the power design can be applied to instances of any logic module.

If this option is specified, the power design can only be applied to instances of a logic module whose name exactly matches one of the specified names.

The module names may not 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 input ports in a module to which this power design applies.

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 deprecated in a future release.

power_design_name

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 model 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:** When specified, the ports of the DUT (design under test) 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.

Example

An 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 ...
```

However, the following generates an error because it is an error to apply the power design to `I1` since the logic module of `I1` is `foo_1`, not `foo1`.

```
set_design my_power_design -modules { foo1 }
set_instance I1 -design my_power_design ...
```

Related Information

Power Design on page 38

end_design on page 173

find_design_objects on page 176

get_parameter on page 180

set_instance on page 202

update_design on page 227

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`*

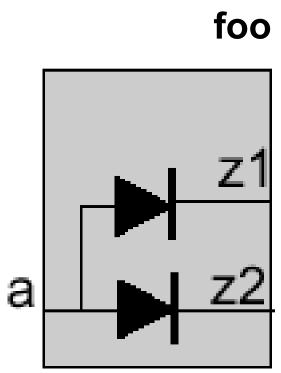
Specifies the names of the ports that connect to the negative inputs or cathode of a diode.

`-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_model 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 208

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` *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
`-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, 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 negation of `pcm/A` as the control driver.

  ```
  set_equivalent_control_pins -master pcm/A -pins {!pcm/B pcm/C} -rules myISO
  ```

Related Information

create_isolation_rule on page 129

create_power_domain on page 150

create_state_retention_rule on page 165

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.

Related Information

end_macro_model on page 174

set_input_voltage_tolerance on page 199

set_macro_model on page 208

set_wire_feedthrough_ports on page 226

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 Separator on page 47

Information Inheritance on page 106

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.

Options and Arguments

`-domain` *`power_domain`*

Indicates that the specified voltage tolerance applies to the input pins of the specified power domain.

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 to which the voltage tolerance applies.

Note: If neither this option, nor the `-domain` option are specified, 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 266

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. (Scope changes are discussed in CPF Modeling for Hierarchical Design on page 72.)

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 below:

```
set_cpf_version
set_array_naming_style
set_hierarchy_separator
set_register_naming_style
define_library_set
define_global_cell
define_isolation_cell
define_level_shifter_cell
define_open_source_input_pin
define_power_clamp_cell
define_power_clamp_pins
define_power_switch_cell
define_state_retention_cell
define_related_power_pins
set_power_unit
set_time_unit
```
`set_instance` (only allowed if used to determine scope; i.e., there are no options or arguments)

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*.

Important

Any rule created in the block-level CPF file that references a virtual port that is declared using the `-input_ports`, `-output_ports`, or `-inout_ports` options 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`*

Links a previously defined power design to the specified instance. The scope is not changed when this option is used.

A *bound* power design is a power design associated with

- an instance through the `set_instance` command, or
- the top level module of a design

To bind a power design to the top level module of a design,

- a user can specify the power design name at the command level when reading or elaborating the CPF; or
- 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`*}

`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 that a previously loaded CPF description is to be used 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.

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.

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 `inst1_foo` and `inst2_foo` are instantiations of the same logic module `foo` under top level module `top`.
  - 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
```

- each instance uses a different power design

```
set_design foo_on
...
end_design foo_on
set_design foo_off
...
end_design foo_off
set_design top
set_instance inst1_foo -design foo_on
set_instance inst2_foo -design foo_off
end_design top
```

- In the following example, '`set_instance inst1_foo`' will change the scope to `inst1_foo`. As a result, the power design `foo` is only defined for scope `inst1_foo`, and it is an error in the second `set_instance` command since the scope at this point is at the top level.

```
set_design top
...
set_instance inst1_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.

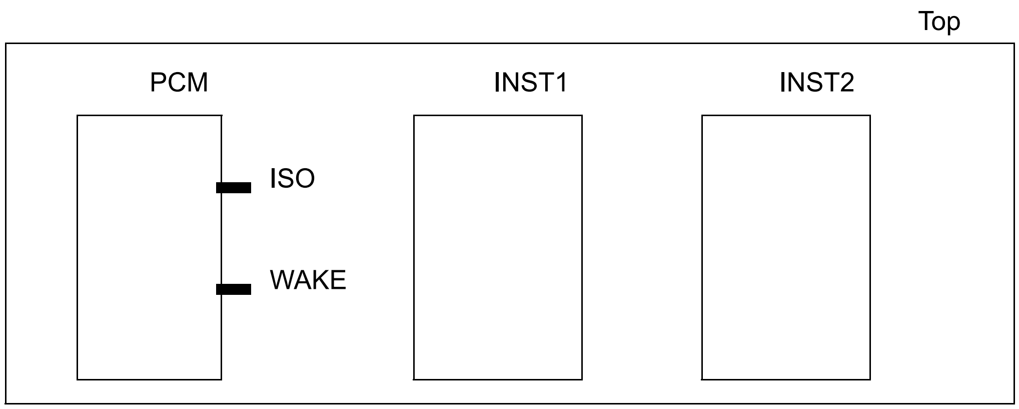
  Design `Top` has three hierarchical instances:

  - `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 -input_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
```

Related Information

Design Objects on page 34

Power Design on page 38

CPF Syntax on page 43

CPF Modeling for Hierarchical Design on page 72

set_design on page 190

set_macro_model

```
set_macro_model macro_model_name [-cells -cell_name_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 the following CPF commands are allowed in a macro model definition; all others are disallowed:

```
create_isolation_rule
create_mode_transition
create_nominal_condition
create_power_domain
create_power_mode
create_power_switch_rule
create_state_retention_rule
define_open_source_input_pin
set_analog_ports
set_diode_ports
set_floating_ports
set_input_voltage_tolerance
set_pad_ports
set_power_source_reference_pin
set_wire_feedthrough_ports
update_power_domain
```

Note: 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.

Important

Within a macro model definition, you cannot have another CPF model.

Options and Arguments

`-cells` *`cell_name_list`*

This option lists the names of the cells to which the macro model applies. When this option is specified, 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 the user 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 may not 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 example models a RAM cell.

Assume the 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 shutoff externally. Power pin `VDD` drives the rest of the peripheral logic of the memory. When `VDDA` is on, `VDD` has to be on. The RAM cell may be described as follows:

```
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 may 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 RAMINST -model 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

end_macro_model on page 174

set_floating_ports on page 197

set_input_voltage_tolerance on page 199

set_wire_feedthrough_ports on page 226

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 118

set_macro_model on page 208

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 (See Power Source Domain on page 39).

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

set_macro_model on page 208

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.
`-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.

Related Information

end_power_mode_control_group on page 175

Power Mode Control Groups on page 84

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 215

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 106

set_power_target on page 214

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.

Note: This command is only needed to specify RTL-to-netlist name conversion, and only applies if the design is RTL.

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 48.

Information Inheritance on page 106

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 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 scope. Targets are only selected from these instances.
- When you specify only the `-instances` option, targets are only selected from the specified hierarchical 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 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 specified with the `-instances` option. Targets are only selected from these instances.
- 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 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.

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.

  The `-type` option is mandatory when this action is specified.

  Note: This action 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
- `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.

`-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 initial blocks must be selected.

Instance names can contain wildcards.

`-lib_cells` *`lib_cell_list`*

Specifies a list of library cells. Targets are selected from all instances of these cells within the selected scope.

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 by the `-instances` or the `-domains` options.

`-targets` *`target_list`*

Specifies a list of object names in the design. If the type is an initial block, it is referenced by the label of the statement inside.

The object name can be a hierarchical name. For example, `a.b.c.thisBlock`.

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.

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 | wreal | 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.

Note: This option is mandatory with the `disable_corruption` action and cannot be used with any other action.

Note: A type of `module` or `instance` may cause mismatch between simulation verification and implementation.

Examples

- The following command selects all initial blocks 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 modules 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 instances of modules `M1` and `M2`.

  ```
  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
  ```

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

  This command can be used to specify the immediate action to be taken when the power is restored to power domains.

  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 -action power_up_replay
    ```

- Case 2

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

  ```
  set_sim_control -action power_up_replay -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 -action power_up_replay
  ```

- Case 3:

  Consider the following block level command:

  ```
  set_sim_control -target initial1 -instances {i1 i2 ...} \
       -action power_up_replay
  ```

  The top-level equivalent command will be:

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

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 that the toggle rate is to be computed 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 106

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.

Related Information

end_macro_model on page 174

set_floating_ports on page 197

set_input_voltage_tolerance on page 199

set_macro_model on page 208

update_design

```
update_design
     power_design_name
```

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 (see Command Categories on page 102)

Options and Arguments

power_design_name	Specifies the name of a previously declared power design.

Related Information

set_design on page 190

end_design on page 173

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 shall be an error if the function of a specified cell conflicts with the expected isolation output specified by the `-isolation_output` option in `create_isolation_rule`. **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`*

If there is a CPF macro model with the same name as the first name specified in `-cells`, then this option can be used to specify the mapping of the domains in the macro model with the top level domains.

The format of each domain mapping is

`{`*`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.

`-location {to|from|parent|any}`

Specifies where to insert the isolation logic.

- `to` – instantiates 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.
- `from` – instantiates 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` – instantiates the isolation logic in the logic hierarchy as determined below:
  - 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
- `any` – works only with the `-within_hierarchy` option to indicate that the isolation logic may be instantiated in any power domain to which the specified logic hierarchy belongs.

Default: `to`

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` 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.

For more information on how implementation tools shall insert the isolation logic, see Isolation Insertion on page 95.

`-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 mappings, where the format of each mapping is

{*`cell_pin design_pin_reference`*}

- *`cell_pin`* is the name of a pin of the first cell specified in the option `-cells`.
- *`design_pin_reference`* is one of the following:
  - the name of a design port or an instance pin. The "!" character is prepended 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`

	This option shall not refer to either the data input port or the data output port of the first cell specified by `-cells`. There must be one and only one of the model's output ports that is not specified in the pin mapping, and this port must be connected to the signal that is driven by the signal being isolated. There must be one and only one of the cell's input ports that is not specified in the pin mapping, and this port must be connected to 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`	When this option is specified, a simulation tool shall not apply the default isolation logic based on the `create_isolation_rule` semantics. Instead, it will use the functional model of the first cell specified in `-cells`.
`-within_hierarchy` *`instance`*	Specifies the instance under which the isolation logic (with or without a hierarchy wrapper) is to be inserted. Use a single hierarchical separator if the logic is 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_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.

Example

The folllowing example specifies that for isolation rule `foo`, isolation cells `cell1` or `cell2` must be used.

```
update_isolation_rules -name foo -cells {cell1 cell2}
```

Assume that the enable pin of `cell1` is `ISO`. The following specifies that the simulation model of `cell1` must be used for the simulation of isolation logic associated with this rule, and the

`-pin_mapping` option specifies that the enable pin must be connected to an instance pin with path name of `pcm/iso_en`.

```
update_isolation_rules -name foo -cells {cell1 cell2} -use_model \
    -pin_mapping {{ISO pcm/iso_en}}
```

Related Information

Isolation Cell on page 40

Isolation Rules on page 94

Isolation Insertion on page 95

create_isolation_rule on page 129

identify_power_logic on page 182

define_isolation_cell on page 260

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_lists`*

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 level shifter cells defined with the `define_level_shifter_cell` command.

`-location {to|from|parent|any}`

Specifies where to insert the level shifters.

- `from` – instantiates the level shifters 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.
- `to` – instantiates 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` – instantiates the level shifters in the logic hierarchy as determined below:
  - 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
- `any` – works only with the `-within_hierarchy` option to indicate that the level shifter may be instantiated in any power domain to which the specified logic hierarchy belongs

Default: `to`

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.

For more information on how implementation tools shall insert the level shifter logic, see Level Shifter Insertion on page 92.

Important

This option is ignored in the case that 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 the level shifter 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.

`-within_hierarchy` *`instance`*

Specifies the instance under which the level shifters (with or without a hierarchy wrapper) are to be inserted.

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.

Important

This option is not allowed with multi-stage level-shifting.

Example

The following example specifies that for level shifter rule `foo`, level shifter cells `cell1` or `cell2` must be used and they must be placed in the originating domain.

```
update_level_shifter_rules -name foo -cells {cell1 cell2} -location from
```

Related Information

Level Shifter Cell on page 40

Level Shifter Insertion on page 92

create_level_shifter_rule on page 134

define_level_shifter_cell on page 266

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 `-power_library_set` 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 in `-library_set` will be used for power analysis and optimization.

Related Information

create_nominal_condition on page 142

define_library_set on page 172

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.

A power domain may be implemented into multiple disjoint physical regions, where each region has its own power and ground nets. In this case, the `-equivalent_power_nets` and `-equivalent_ground_nets` options may be used. For equivalent power and ground nets

- 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 that is 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`, then domain `X`'s equivalent nets 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` *`port_list`*

Incrementally updates a power domain's boundary port membership with the specified list. The effect of this option will be the same as the `-boundary_ports` option in the `create_power_domain` command.

This option is useful when there are additional boundary ports created during implementation that may not be assigned to the correct power domain.

`-deep_nwell_net` *`net`*

Specifies the net to be used for the deep nwell connection. This net must have been declared using the `create_bias_net` command.

`-deep_nwell_net` *`net`*

Specifies the net to be used for the deep pwell connection. This net must have been declared using 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

Wildcards are allowed.

`-equivalent_power_nets` *`power_net_list`*

Specifies a set of power nets that are equivalent to the primary power net of the power domain

Wildcards are allowed.

`-instances` *`instance_list`*

Incrementally updates a power domain's instance membership with the specified list. The effect of this option will be the same as the `-instances` option in the `create_power_domain` command.

This option can be used to ensure the correct power domain assignment for instances created during implementation.

For example, top-level buffering during physical implementation may introduce buffers at 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.

`-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.

`-name` *`domain`* Specifies the name of the power domain.

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

`-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.

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

In the following example, the command specifies that the primary power and ground nets for the power domain are `VDD` and `VSS` respectively. In addition, this power domain may be implemented with multiple physical partitions where each partition may have its own local power and ground net names. In this case, in addition to the primary power and ground nets `VDD` and `VSS`, power nets `VDD1` and `VDD2` and ground nets `VSS1` and `VSS2` should be used for the power and ground connections within partitions.

```
update_power_domain -name foo -primary_power_net VDD -primary_ground_net VSS \
    -equivalent_power_nets {VDD1 VDD2} -equivalent_ground_nets {VSS1 VSS2}
```

Related Information

create_bias_net on page 123

create_ground_nets on page 127

create_power_domain on page 150

create_power_nets on page 161

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
     | -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 volts (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.

`-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.

It is an error if `-hold_sdc_files` is specified in combination with `-sdc_files`.

`-hold_sdc_files` may be specified in combination with `-setup_sdc_files`.

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 volts (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.

`-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 constraints for both late and early analysis.

It is an error if `-sdc_files` is specified in combination with either or both of `-setup_sdc_files` and `-hold_sdc_files`.

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.

It is an error if `-sdc_files` is specified in combination with `-setup_sdc_files`.

`-setup_sdc_files` may be specified in combination with `-hold_sdc_files`.

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 39

create_power_mode on page 158

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 load pin connected to the power switch cell output pin that is the buffered output of the pin specified in the corresponding `enable_condition_`*`x`*

The expression must be unary.

The expression is a function of an input pin of the power controller.

Note: If a switch rule has `-acknowledge_receiver_2` specified, it must also have `-acknowledge_receiver_1` specified.

`-average_ir_drop_limit` *`float`*

Specifies the maximum allowed average voltage change across a power switch due to resistive effects in volts (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_1` 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.

Note: If a switch rule has `-enable_condition_2` specified, it must also have `-enable_condition_1` specified.

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

`-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 volts (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 127

create_power_nets on page 161

create_power_switch_rule on page 163

update_state_retention_rules

```
update_state_retention_rules
     -names rule_list
     { -cell_type string
     | -set_reset_control
     | -cells cell_list [-use_model -pin_mapping pin_mapping_list
        [-domain_mapping domain_mapping_list]] }...
```

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 not 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 that was 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 which all have the same retention behavior.

If you specify this option with the `-cells` options, the `-cells` option takes precedence.

`-domain_mapping` *`domain_mapping_list`*

If there is a CPF macro model with the same name as the name of the first cell, then this option can be used to specify the mapping of the domains in the macro model with the top level domains.

The format of each domain mapping is

{*`domain_in_child_scope domain_in_parent_level_scope`*}

The specified domain mapping applies to all instantiations of the specified retention cell.

It is an error if a macro model is specified and the macro model has multiple domains but this option is not specified.

`-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 mappings, where the format of each mapping is

{*`cell_pin design_pin_reference`*}

- *`cell_pin`* is the name of a pin of the first cell specified in the option `-cells`.
- *`design_pin_reference`* is one of the following:
- the name of a a design port or an instance pin. The "!" character is prepended 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`
  - `restore_signal`, which refers to the expression in `-restore_edge` or `restore_level`

`-set_reset_control`	Indicates that the rule is intended for 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).
`-use_model`	When this option is specified, a simulation tool shall not apply the default state retention logic based on the `create_state_retention_rule` semantics. Instead, it will use the functional model of the first cell specified in `-cells`.

Example

The following example specifies that for state retention rule `foo`, state retention cells `cell1` or `cell2` must be used.

```
update_state_retention_rules -name foo -cells {cell1 cell2}
```

Assume that the restore pin of `cell1` is `RESTOREN`. The following specifies that the simulation model of `cell1` must be used for simulation of the retention logic associated with this rule, and the `-pin_mapping` option specifies that the restore pin must be connected to an instance pin with a path name of `pcm/ret_en` with an inverter.

```
update_state_retention_rules -name foo -cells {cell1 cell2} -use_model \
    -pin_mapping{{RESTOREN !pcm/ret_en}}
```

Related Information

State Retention Cell on page 41

create_state_retention_rule on page 165

define_state_retention_cell on page 286

10

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 ]
```

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.

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 40

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 ] ]
```

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

Examples

- 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
```

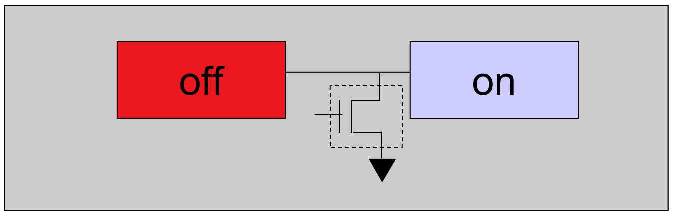
define_isolation_cell

```
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]
```

Identifies the library cells in the .lib files that must be used as isolation cells.

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.

An isolation cell may be of a clamp-type, as shown below.



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 a specified pin is related to the

- non-switchable supply
  - the pin must also be specified in the `-always_on_pins` option
  - the logic that drives this pin must be on when the isolation enable is asserted
- switchable supply
  - the logic that drives this pin can be corrupted when the isolation enable (specified by the `-enable` option) is asserted
  - the specified pin needs to be set to the controlling value of the gate before isolation enable is asserted or deasserted.

`-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
- one optional enable pin that applies to the specified path

The format of each group in the list is

{ *`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` – indicating that this type of cell may 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 and ground pins for normal function and relies on secondary power or ground pins to provide the isolation function.
- `to` – indicating 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. This type of cell may only be used for `off` to `on` isolation.
- `on` – indicating 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 may 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` – this option is redundant with the `any` option and will be deprecated a future release.

- `any` – indicating that this type of cell can still provide its normal function and isolation function 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 may 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} }
```

- The following example illustrates how to model an isolation cell that must be placed in an unswitched domain.

```
define_isolation_cell -cells cell1
    -power VDD -ground VSS \
    -valid_location on -enable ISO
```

- The following example illustrates a special isolation cell that has two enable pins and can be placed in a switchable power domain. One of the enable pins `EN` is related to the switchable supply `VDDS`, and the other enable pin `ISO` is related to the non-switchable supply `VDD`.

  ```
  define_isolation_cell -cells cell2
      -power_switchable VDDS -power VDD -ground VSS \
      -valid_location from -enable ISO -aux_enables EN
  ```

Related Information

Isolation Cell on page 40

Information Precedence on page 105

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 group_list]
     [-valid_location {to | from | either | any}]
     [-bypass_enable expression] [-multi_stage integer]
```

Identifies the library cells in the .lib files that must be used as level shifter cells

Important

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.

`-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 element of the list specifies a group of

- one cell input pin
- one cell output pin
- one optional enable pin that applies to the specified path

The format of each group in the list is
`{`*`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` can be different 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 \
-input_power_pin VH -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 up \
-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 40

Information Precedence on page 105

create_level_shifter_rule on page 134

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. The command can be used 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 as 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.

Example

Assume you have an IO cell with the following Verilog model:

```
module IC33 (Z, PAD);
  input PAD;     // related to DVDD & DVSS
  output Z       // related to VDD & GND
  assign Z = PAD;
endmodule
```

The following is how to model it in CPF for the cell and a possible usage in the design when the cell should be used for ports `IN1` and `IN2`, the core domain is `PD_VDD`, and the IO domain is `PD_VDD330`:

```
define_pad_cell -cells IC33 -pad_pins PAD \
    -pin_groups { {core_group Z} {pad_group PAD}}
define_related_power_pins -data_pins Z -cells IC33 -power VDD -ground GND
define_related_power_pins -data_pins PAD -cells IC33 -power DVDD -ground DVSS

create_pad_rule -name pad1 -of_bound_ports "IN1 IN2" \
    -mapping { {core_group PD_VDD} {pad_group PD_VDD330}}
```

Related Information

create_pad_rule on page 148

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 ]
```

Identifies the library cells in the .lib files that must be used as power switch cells.

The input enable and output enable pins of this cell are related to the non-switchable power and ground pins.

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_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.

`-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 ohms.

`-stage_1_output (-stage_2_output)` *`expression`*

Specifies whether the output pin specified in the expression is the buffered or inverted output of the input pin specified through the corresponding -`stage_`*`x`*`_enable` option.

The pin specified through the -`acknowledge_receiver` option of the `create_power_switch_rule` command is connected to the output pin specified through

- The `-stage_1_output` option if the `-stage_2_output` option is omitted.
- The -`stage_2_output` option if both -`stage_1_output` and `stage_2_output` options are specified.

Note: If neither option is specified, the pin specified through the -`acknowledge_receiver` is left unconnected.

`-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 41

Expressions on page 51

Information Precedence on page 105

create_power_switch_rule on page 163

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 ]
```

Identifies the library cells in the .lib files that must be used as state retention cells.

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 can 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` *`expression`*

Specifies an additional condition that must be met (after the primary power domain of the retention cell is shut off and before the domain is powered on again) for the retention operation to be successful.

The expression 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"
```

Related Information

State Retention Cell on page 41

Information Precedence on page 105

A

Quick Reference

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

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 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}...
     [-exclude pin_list]
     [-isolation_target {from|to}]
     [-isolation_output { high | low  | hold | tristate | clamp_high | clamp_low}]
     [-isolation_control list_of_additional_controls]
     [-secondary_domain power_domain]

create_level_shifter_rule
     -name string
     {-pins pin_list | -from power_domain_list | -to power_domain_list}...
     [-force]
     [-exclude pin_list]
     [-bypass_condition expression]
     [-output_domain power_domain] [-input_domain power_domain]

create_mode
     -name string -condition expr
     [-probability float] [-illegal]

create_mode_transition
     -name string
     -from mode -to 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 {high|low|random|inverted} ]
     [-power_down_states {high|low|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 group_list]
     [-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_model_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_name [-modules module_name_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_name_list]

set_pad_ports pin_list

set_power_source_reference_pin pin
     -domain power_domain -voltage_range min:max
```

```
set_power_mode_control_group -name group
     { -domains domain_list
     | -groups group_list
     | -domains domain_list -groups group_list}

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
     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
     | -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
     | -set_reset_control
     | -cells cell_list [-use_model -pin_mapping pin_mapping_list
        [-domain_mapping domain_mapping_list]] }...
```

B

Chapter by Chapter Summary of Changes

CPF 1.1 to CPF 2.0

Changes to section and command content are noted in bulleted lists.

New sections are noted as such beside the name.
Unchanged sections are not listed.

New commands are noted as such beside the command name.
Unchanged commands are not listed.

1 Introducing the Common Power Format

- no changes

2 Terminology

Design Objects

- Followpins: added definition

CPF Objects

- Mode: added definition
- Power Design: added definition
- Power Source Domain: added definition

Special Library Cells for Power Management

- Always On Cell: updated definition
- Global Cell: added definition

3 CPF Syntax

Hierarchy Separator

- added instructions for escaping the separator when the separator is also part of an Object's name

Bus Delimiters

- added examples for using square bracket delimiters

Expressions

- added a note regarding expressions containing an object whose name contains a character that is also a Boolean operator
- added additional operators for Boolean expressions: ( ) ~ ^ && ||

4 CPF Example

- updated to reflect CPF 2.0

5 Power Domains

Primary and Secondary Power Domains

- introduces the identify_secondary_domain command
- added tip regarding the power domain of a special low power cell

Secondary Power Domain of Isolation Instances

- added explanation of semantic differences between multi-stage and single-stage level shifters
- added semantics for `-valid_location` attribute

Secondary Power Domain of Isolation Instances

- updated description, adding some explanation of semantics for simulation, synthesis, physical implementation, and verification tools
- added explanation of semantics for single-rail isolation cells

Secondary Domain of Retention Logic

- updated description, including simulation semantics

Secondary Domain of Global Cells - new section

Power Domain Mapping Concepts

- added tip and example regarding conflicting power domain mapping and assignments

CPF Modeling for Hierarchical Design

- new semantics regarding commands between set_instance and either set_design or set_macro_model
- new semantics for appending power intent to an existing design
- updated examples

6 Power Modes

Generic Mode - new section

An Illegal Domain Configuration - new section

Power Mode Control Groups

- updated description of power mode control group concept

Explicitly Created Power Mode Control Groups

- added example “User-Created Power Control Groups with Domains in Multiple Groups”

7 Precedence and Semantics of CPF Rules

When Can Implementation Tools Ignore a Level Shifter Rule

- section name changed from “When Not to Implement a Level Shifter Rule”
- updated list of conditions to ignore rule

Level Shifter Insertion - new section

How to Handle Isolation Rules Created Without Isolation Condition - new section

When Can Implementation Tools Ignore an Isolation Rule

- section name changed from “When Not to Implement an Isolation Rule”
- updated list of conditions to ignore rule

Isolation Insertion - new section

When Can Implementation Tools Ignore a State Retention Rule

- section name changed from “When Not to Implement a State Retention Rule”

8 CPF Semantics

Referencing Objects

- updated semantics for hierarchical paths

9 General CPF Commands

create_analysis_view

- added `-default` option

create_global_connection

- added `-ports` and `-pg_type` options

create_isolation_rule

- added `-force` and `-isolation_control` options
- added `clamp_high` and `clamp_low` values to `-isolation_output` option
- updated descriptions of `-exclude`, `-isolation_output`, and `-secondary_domain` options

create_level_shifter_rule

- added `-force`, and `-bypass_condition`, `-output_domain`, and `-input_domain` options
- updated description of `-exclude` option

create_mode - new command

create_mode_transition

- added `-assertions` and `-illegal` options
- added generic mode semantics

- added tip regarding mode transitions

create_nominal_condition

- added -deep_pwell_voltage and -deep_nwell_voltage options
- modified options that specify a voltage to take either a single voltage or a list of minimum, nominal, and maximum values
- added instructions and examples for specifying voltages and voltage lists
- updated descriptions of all options that use voltage or voltage_list values
- updated descriptions of -name and -state options

create_operating_corner

- changed string to corner for the value of the -name option
- added -power_library_set option
- updated description of -library_set option and changed its value to a list

create_pad_rule - new command

create_power_domain

- added -exclude_instances, -exclude_ports, -power_down_states, and -power_source options
- added -inverted value to -power_up_states option
- updated descriptions of -boundary_ports, -default_isolation_condition, and -instances options
- added examples

create_power_mode

- added -condition option
- updated command description
- added example

create_state_retention_rule

- added -required, -retention_precondition, and -use_secondary_for_output options
- clarified the semantics of expressions used in options
- updated descriptions of -restore_edge, -restore_level, restore_precondition, save_edge, -save_level, and -save_precondition options

end_design

- changed module option to power_design_name

find_design_objects - new command

identify_secondary_domain

- updated command description

set_analog_ports - new command

set_design

- set_design semantics changed to identify a power design, not a module
- changed *module* option to *power_design_name*
- added -modules, -input_ports, -output_ports, inout_ports, and -testbench options

- updated description for `-ports` option

set_diode_ports - new command

set_equivalent_control_pins

- changed `pin_list` value to `-pin_expression_list` for `-pins` option
- updated descriptions for `-domain` and `-pins` options
- added example

set_input_voltage_tolerance

- removed `-ports` and `-bias` options
- added `-power`, `-ground`, `-domain`, and `-pins` options
- updated command description
- added examples

set_instance

- `-design` option now refers to a power design instead of a module
- changed semantics regarding allowed commands between set_instance and either set_design or set_macro_model
- updated description for `-port_mapping` option
- added examples

set_macro_model

- added new commands to list of commands allowed in a macro model definition
- added note regarding non power and ground ports
- added `-cells` option

set_pad_ports - new command

set_power_source_reference_pin - new command

set_sim_control - new command

update_design - new command

update_isolation_rules

- added `-suffix`, `-use_model`, `-pin_mapping`, and `-domain_mapping` options
- added `-parent` and `-any` values to `-location` option
- updated descriptions for `-location` and `-within_hierarchy` options

update_level_shifter_rules

- added `-through` and `-suffix` options
- added `-parent` and `-any` values to `-location` option
- added `-list_of_cell_lists` value to `-cells` option
- updated descriptions for `-location` and `-within_hierarchy` options

update_nominal_condition

- added `-power_library_set` option
- updated description for `-library_set` option

update_power_domain

- added `-instances`, `-boundary_ports`, `-deep_nwell_net`, and `-deep_pwell_net` options
- updated note regarding the specification and combination of options
- updated descriptions for options `-nmos_bias_net`, `-pmos_bias_net`, `-primary_ground_net`, and `-primary_power_net`

update_state_retention_rules

- added `-use_model`, `-pin_mapping`, and `-domain_mapping` options

10 Library Cell-Related CPF Commands

define_global_cell - new command

define_isolation_cell

- changed syntax for `-power_switchable`, `-ground_switchable`, `-power`, and `-ground` options
- added `-clamp`, `-pin_groups`, and `-aux_enables` options
- added `-any` value to `-valid_location` option
- updated command description
- added examples

define_level_shifter_cell

- added `-pin_groups`, `-bypass_enable`, and `-multi_stage` options
- added `-any` value to `-valid_location` option
- changed all `voltage` values to `voltage_list`
- changed all `voltage_range` values to `voltage_range_list`
- added semantic clarifications to command description
- changed descriptions of all options that use `voltage_range_list` and `voltage_range_list` values
- changed semantic note for `-input_ground_pin`, `-input_power_pin`, `-output_ground_pin`, and `-output_power_pin` options
- added examples

define_open_source_input_pin

- added `-type` option

define_pad_cell - new command

define_power_clamp_pins - new command

define_power_switch_cell

- added semantic clarifications to command description

define_related_power_pins

- added semantic note to command description

define_state_retention_cell

- added `-retention_check` option
- added semantic note to command description
- updated descriptions of `-restore_check`, `-restore_function`, `-save_check`, and `-save_function` options

Errata as of 15-November-2011 for Si2 Common Format Specification V2.0

Text additions are in blue. Text to be ignored or deleted is colored red, optionally with strikethrough if it is mixed with other text.

Contents

Document Content Errors

Chapter "Enhancements and New Capabilities for CPF 2.0", p 14

The subsection "Added Capability to Find Design Objects within a Specified Scope" should be in the "Other Improvements" section on page 22.

Chapter "Enhancements and New Capabilities for CPF 2.0", p 22

The section heading "Simulator-Related Changes" should be "Simulation-Related Changes".

Chapter 3 "CPF Syntax", p 49

In the "Specifying the Representation of the Bits" section, `set_analog_ports` should be `set_array_naming_style`:

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_analog_ports`~~ `set_array_naming_style` command.

Chapter 5 "Power Domains", p 70

The descriptions of the examples presented at the top of the page incorrectly refer to the "macro instance" when it should be simply "instance". The section with cross-out corrections is shown below:

```
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

Chapter 9 “General CPF Commands”, p 171

There is a typo in the option name `–use_secondary_for_putput` at the top of the page. The correct option name is:

```
-use_secondary_for_output
```

Chapter 9 “General CPF Commands”, p 208
Appendix A “Quick Reference”, p. 297

The following syntax for `set_macro_model` contains an error.

```
set_macro_model macro_model_name [-cells -cell_name_list]
```

The value to option `–cells` should be *`cell_name_list`* without a preceding dash “-“. The correct syntax is as follows:

```
set_macro_model macro_model_name [-cells cell_name_list]
```

Semantic Clarifications

Chapter 4 “Power Domains”, pp 72-76

The CPF 2.0 specification allows the nesting of power design definitions, as shown in examples 5-1 through 5-9 in the specification document. This could lead to legal but complicated nesting of power design definitions, possibly as a result of CPF files including other CPF files. For example, the following power design named `chip` contains one instance of power design `A`, which nests power designs `B`, `C`, and `D`, and directly instantiates power designs `E` and `B`:

```
set_design chip
    set_instance instA
    set_design A
         set_design B
               set_design C
                     set_design D
                     end_design D
               end_design C
         end_design B

         set_instance instE
         set_design E
               set_instance instF
               set_design F
                   set_instance instD -design D   ; # ← .instA.instE.instF.instD
               end_design F
         end_design E
         set_instance instB -design B            ; # ← .instA.instB
    end_design A
end_design chip
```

In this example, power design definitions for `B`, `C`, and `D` are nested within the definition of `A`, and all belong to the scope of `.instA` (only `set_instance` changes scope, `set_design` does not.)

Before power design `A` is instantiated, power designs `A`, `B`, `C`, and `D` are all considered to be defined in the top, or global, scope, and can be referenced in any descendant scope as long as the reference is not recursive. To extend this concept, once `A` is instantiated these same power designs are no longer global but can still be referenced in a descendant scope. This rightly implies that `.instA.instE.instF.instD` is a legal reference to power design `D` even though `D`’s definition is nested within other power design definitions.

The instance `.instA.instB` references power design `B` in the same scope. Power design `C` is ignored because it is not bound to any instances.

▶ **Tip:** For best practice and readability, it is recommended to avoid nesting power design definitions that do not follow `set_instance`. Individual, "un-nested" power designs can be defined in the global scope and referenced later.

▶ **Warning:** The ability to nest power design definitions may be deprecated in a future release.

Two equivalent but non-nesting implementations of the previous example are presented below. The first is a mixture of techniques that instantiates and defines power designs inline (i.e. following `set_instance`) but also instantiates previously defined designs, and the second only instantiates previously defined power designs.

1) Power designs `B`, `C`, and `D` are defined at the global scope, and `A`, `E`, and `F` are instantiated and defined inline, following `set_instance` as before:

```
set_design B
end_design B

set_design C
end_design C

set_design D
end_design D

set_design chip
    set_instance instA
    set_design A
         set_instance instE
         set_design E
               set_instance instF
               set_design F
                   set_instance instD -design D   ; # ← .instA.instE.instF.instD
               end_design F
         end_design E
         set_instance instB -design B             ; # ← .instA.instB
    end_design A
end_design chip
```

The disadvantage of instantiating definitions of power designs inline is that each definition can be used only used once. For example, power design `F` can be bound to only one instance, `instF`. If power design `F` is required again, its definition must be repeated. If the overall hierarchical design does not require multiple instantiations of a particular power design, this is not an issue.

2) All power designs are defined at the global scope and are referenced with "`set_instance –design`"

```
set_design B
end_design B
```

```
set_design C
end_design C

set_design D
end_design D

set_design F
    set_instance instD -design D
end_design F

set_design E
    set_instance instF –design F
end_design E

set_design A
    set_instance instE –design E
    set_instance instB -design B
end_design A


set_design chip
    set_instance instA –design A
end_design chip
```

This technique allows a single power design definition to be bound to multiple instances.

Chapter 4 "Power Domains", p 73
Chapter 9 "General CPF Commands", p. 202.

On page 73, the paragraphs introducing the `update_design` command do not specify how to append power intent to a power design definition that follows a `set_instance` command. The following illustrates a solution that is legal in CPF 2.0:

```
set_instance instA
set_design A
…
end_design A

set_instance instA
update_design A   ; # ← this appends power intent to A
…
end_design A
```

The updated definition of A in scope `instA` is appended to the first definition of A in scope `instA`.

According to the semantics of CPF 2.0, if the `update_design` is changed to a `set_design`, the contents of the second definition are ignored.

```
set_instance instA
set_design A
…
end_design A

set_instance instA
set_design A     ; # ← this definition is ignored in CPF2.0
…
end_design A
```

The preceding example was legal in CPF1.1. If this technique was used in a CPF1.1 file to append power intent to a power design, the second `set_design` must be changed to `update_design` to be compliant with CPF 2.0.

On page 202, the third paragraph in the description of `set_instance` requires the following modifications:

> 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`, `update_design`, ~~command~~ or ~~a~~ `set_macro_model` command. Some commands are allowed between `set_instance` and `set_design`, `update_design`, or `set_macro_model`. These commands are shown below:
> *…etc…*

Errors in Example Code

Chapter 4 “Example”, p 56

Power mode `PM1` should be specified as the default power mode for power design `top`.

```
create_power_mode -name PM1 -domain_conditions {PD1@high PD2@medium PD3@high \
PD4@low} -default
```

Chapter 5 “Power Domains”, p 74

1. In example 5-4, the second `set_design` statement requires a “`;`” before the comment character:

```
set_design foo ; # second definition of model foo
```

2. In example 5-5, the `update_design` statement requires a “`;`” before the comment character:

```
update_design foo ; # appended definition of model foo
```

Chapter 6 “Power Modes”, p 80

1. The `–conditions` option in the following generic mode command example is incorrect. It should be `-condition`, without an “s” at the end.

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

2. The `–instance` option in the following command from the multi-line example is incorrect. It should be `-instances`, with an “s” at the end.

```
create_power_domain -name MULTIMEDIA -instances "audio_mac video_mac"
```

Chapter 6 “Power Modes”, p 85

In power design BLOCKA, power domain `PD_C1` should be declared as the default power domain:

```
create_power_domain -name PD_C1 -default
```

In power design `top`, power domain `PDB` should be declared as the default power domain:

```
create_power_domain -name PD_B  -default
```

Chapter 6 “Power Modes”, pp 87-88

The two power mode control groups illustrated in Example 6-5 each requires a default power mode.

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

Chapter 9 “General CPF Commands”, p 156

1. There is a line-wrapping error in the command example after the following bullet, where the “`-`“ preceding the `-active_state_conditions` option is orphaned on the previous line. The command requires a backslash for line continuation. The corrected example is below:

   - 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}}
   ```

2. The following bullet and code example illustrate the `–exclude_instances` option for `create_power_domain,` but the option is mistakenly written as `–exclude`:

   - 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_instances` 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_instances {A2 A1/B} ...
   ```

Chapter 9 “General CPF Commands”, p 178

There is an error in the fifth `find_design_objects` command in the example. The `–type` option should be `-object`:

```
find_design_objects a ~~–type~~ -object inst -hierarchical
-> returns 'a' and  'a.a'
```

Chapter 9 “General CPF Commands”, p 194

The `end_macro_cell` statement in the example should be `end_macro_model`:

```
set_macro_model foo
...
set_diode_ports -positive a -negative {z1 z2}
...
~~end_macro_cell~~ end_macro_model
```

Chapter 9 “General CPF Commands”, p 206

There is a typo at the beginning of mod1.cpf. In the `set_design` statement, the list for `-input_ports` is terminated by a right parenthesis whereas it should be a curly brace:

```
set_design mod1 -input_ports { iso_en restore ~~)~~ }
```

Chapter 9 “General CPF Commands”, p 212

In the `set_power_source_reference_pin` statement in the example, the `-voltage_range` option is missing its leading “`-`“:

```
set_power_source_reference_pin AVDD -domain PDVOUT -voltage_range 1.0:1.1
```

Chapter 9 “General CPF Commands”, pp 221-222

1. Throughout the examples,
   - the option `–target` should be `–targets` with an “s” in all `set_sim_control` commands that specify this option incorrectly.
   - the option `–domain` should be `–domains` with an “s” in all `set_sim_control` commands that specify this option incorrectly:

```
set_sim_control -targets {L*} -exclude {L1*} -action power_up_replay

set_sim_control -targets L1 -modules {M1 M2} -action power_up_replay

set_sim_control -targets L -instances {A.B.C} -action power_up_replay

set_sim_control -targets initial1 -action power_up_replay
set_sim_control -targets initial1 -instances foo -action power_up_replay

set_sim_control -action power_up_replay -targets initial1 -domains X

set_sim_control -targets initial1 -domains foo/X -action power_up_replay

set_sim_control -targets initial1 -instances {i1 i2 ...} \
     -action power_up_replay

set_sim_control -targets initial1 -instances {foo/i1 foo/i2 ...} \
     -action power_up_replay
```

2. The first `set_sim_control` statement in Case 1 is missing the required option `–action`.

Case 1

Consider the following block level command in block `foo`:
`set_sim_control -target initial1 –action power_up_replay`

Chapter 9 "General CPF Commands", pp 231-232

Option `–name` should be `–names` with an "s" in both `update_isolation_rules` command examples:

```
update_isolation_rules -names foo -cells {cell1 cell2}

update_isolation_rules -names foo -cells {cell1 cell2} -use_model \
   -pin_mapping {{ISO pcm/iso_en}}
```

Chapter 9 "General CPF Commands", p 236

Option `–name` should be `–names` with an "s" in the `update_level_shifter_rules` command example:

```
update_level_shifter_rules -names foo -cells {cell1 cell2} -location from
```

Chapter 9 "General CPF Commands", p 253

Option `–name` should be `–names` with an "s" in both `update_state_retention_rules` command examples. In addition, in the second example, a space is needed between `–pin_mapping` and `{`:

```
update_state_retention_rules -names foo -cells {cell1 cell2}

update_state_retention_rules -names foo -cells {cell1 cell2} -use_model \
   -pin_mapping {{RESTOREN !pcm/ret_en}}
```

Chapter 10 “Library Cell-Related CPF Commands”, p 272

In the example under the first bullet, the `-input_power_pin` option should be `-output_power_pin`:

```
define_level_shifter_cell \
-cells LSHL* \
-input_voltage_range 1.0 -output_voltage_range 0.8 \
-direction down \
~~-input_power_pin~~ -output_power_pin VH -ground G
```

In the example under the third bullet,

- the fourth line of the command is missing the backslash continuation character at the end of the line
- `groun_input_voltage_range` should be `ground_input_voltage_range`

```
define_level_shifter_cell \
-cells LSLH* \
   -input_voltage_range 0.8 -output_voltage_range 1.0 \
   -ground_input_voltage_range 0.5 -ground_output_voltage_range 0.0 \
   -direction up \
   -input_power_pin VL -output_power_pin VH \
   -input_ground_pin GH -output_ground_pin GL
```

Chapter 10 “Library Cell-Related Commands”, p 277

The option `–of_bound_ports` should be `–of_bond_ports` in the `create_pad_rule` example command near the top of the page:

```
create_pad_rule -name pad1 ~~–of_bound_ports~~ –of_bond_ports "IN1 IN2" \
    -mapping { {core_group PD_VDD} {pad_group PD_VDD330}}
```