cadence®

Common Power Format User Guide

Version 2.0
October 2019

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

Contents

Preface

About This Manual

This document describes how to capture the power intent for your design using the Si2 Common Power Format (CPF), a standardized format for specifying power-saving techniques early in the design process, to deliver an end-to-end low-power design solution to IC engineers.

To use this manual, you should be familiar with IC power consumption concepts.

Additional References

For information on what is new or changed in CPF version 2.0 see *What's New in Common Power Format.*

For reference information about the Common Power Format (CPF), refer to *Common Power Format Language Reference*

The following sources are helpful references, but are not included with the product documentation:

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

Reporting Problems or Errors in Manuals

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

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

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

Customer Support

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

Cadence Online Support

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

For more information on Cadence online support go to:

http://support.cadence.com

Other Support Offerings

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

For more information on these support offerings go to:

http://www.cadence.com/support

Documentation Conventions

The list below describes the syntax conventions used for the Joules commands and attributes.

`literal`	Non italic words indicate keywords that you must type literally. These keywords represent command, attribute 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 }` Braces, used in Tcl command examples, indicate that the braces must be typed in.
`...`	Three dots (...) indicate that you can repeat the previous argument. If the three dots are used with brackets (that is, `[argument]...`), you can specify zero or more arguments. If the three dots are used without brackets (`argument...`), you must specify at least one argument, but can specify more.
#	The pound sign precedes comments in command files.

1

Introduction

In this Guide

This document describes how to capture the power intent for your design using the Si2 Common Power Format (CPF), a standardized format for specifying power-saving techniques early in the design process, to deliver an end-to-end low-power design solution to IC engineers.

Chapter 2, “Creating a CPF File,” shows all the commands needed in a complete CPF file for different advanced low power techniques. For more information about the CPF command syntax, refer to the *Common Power Format Language Reference.*

Note: For the sake of simplicity, the CPF creation described in this document assumes that you start with RTL code that does not contain any instantiations of low power logic.

Chapter 3, “Process of Creating the CPF Content,” shows different approaches to create the CPF content.

Chapter 4, “Hierarchical Flow,” shows the additional constructs you need to create a CPF file for a hierarchical flow.

Chapter 5, “Modeling Special Cells,” shows how to model some of the special power cells needed to support the advanced low power techniques.

Cadence Tools Supporting the Common Power Format

- Conformal Low Power
- Innovus
- Genus
- Modus ATPG
- Tempus Timing Signoff Solution
- Incisive Enterprise Simulator
- Incisive Palladium

Note: For information on the product option, feature, or package that supports CPF, contact your local sales or AE contact.

Refer to the product documentation of the tools for information on

- How CPF is used in the tool
- The tool command(s) related to CPF
- When to read the CPF file in the tool flow
- The CPF commands supported by the tool

2

Creating a CPF File

Introduction

This chapter shows you what the content of a complete CPF file looks like for the following low power techniques:

- Creating a CPF File for an MSV Design
- Creating a CPF File for a Design Using PSO Methodology
- Creating a CPF File for a Design Using DVFS Methodology

Each section is self-contained. If you are interested in only one technique, you will find all the information you need for that low power technique in that section.

This implies that if you intend to use several techniques, you might find some repetition.

This chapter assumes a non-hierarchical flow. For more information about the hierarchical flow, refer to Chapter 4, “Hierarchical Flow.”

For simplicity, only one CPF file is created.

Note: Other chapters will show use of multiple CPF files.

The content of the CPF file can change through the design process. The tools in the design process need different information. Therefore you can start the design with an incomplete CPF file.

Creating a CPF File for an MSV Design

A Multiple Supply Voltage (**MSV**) design uses multiple supply voltages for the core logic. In Figure 2-1 on page 18 the `top` design and instance `inst_A` operate at voltage `VDD1`, while instance `inst_B` operates on voltage `VDD2` and instance `inst_C` operates at voltage `VDD3`.

A portion of the design that operates at the same operating voltage (that is, uses the same *main* power supply) belongs to the ***power domain*** that corresponds to that operating voltage.

A steady state of the design is called a ***power mode***. Pure MSV designs have only one power mode because the operating voltage of the power domains is assumed not to change. A power mode will also have a typical set of timing constraints associated with it.

To pass signals between portions of the design that operate on different voltages, ***level shifters*** are needed.

The majority of cells in a power domain are driven by the same power supply, except for the level shifters which are driven by multiple power supplies.

Level shifters have two sets of power and ground pins, and are therefore associated with two power domains: a ***primary*** and a ***secondary power domain***.

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

The tools reading CPF can derive the primary and secondary power domains for level shifters. For more information, refer to Input and Output Domains of Level Shifters.

Figure 2-1 Example of MSV Design

Operating Voltage (OV)	Instances Operating on OV	Libraries characterized for OV	Power Domains for OV
VDD1: 0.8	top, inst_A	lib1 lib2	PD1
VDD2: 1.0	inst_B	lib3	PD2
VDD3: 1.2	inst_C	lib4	PD3

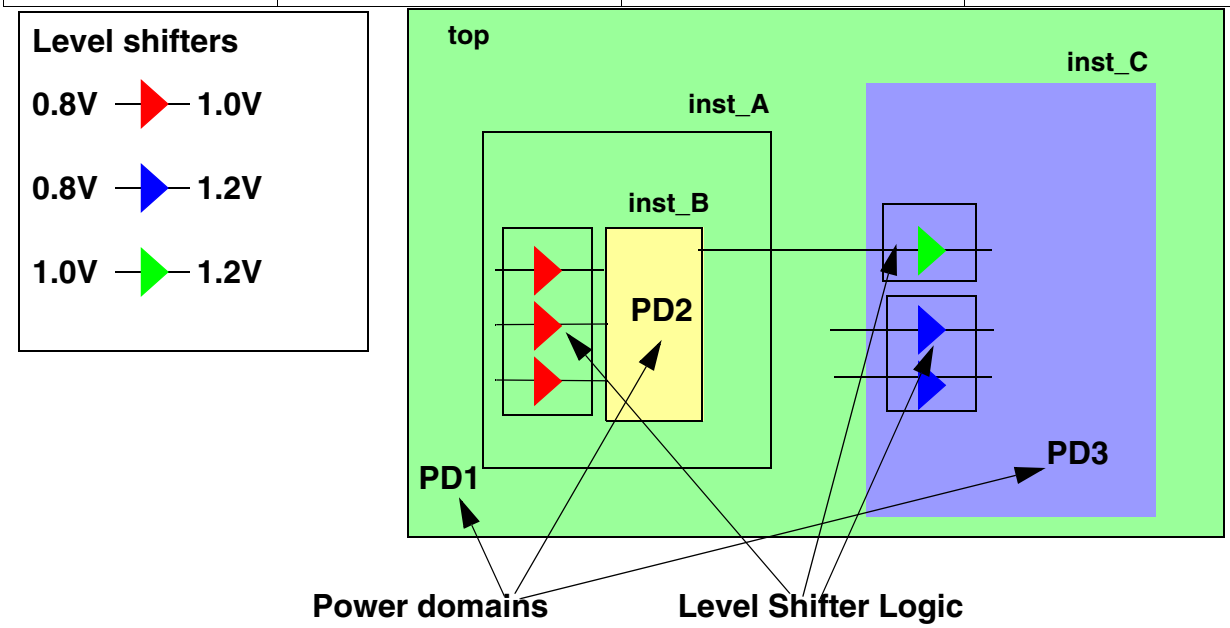
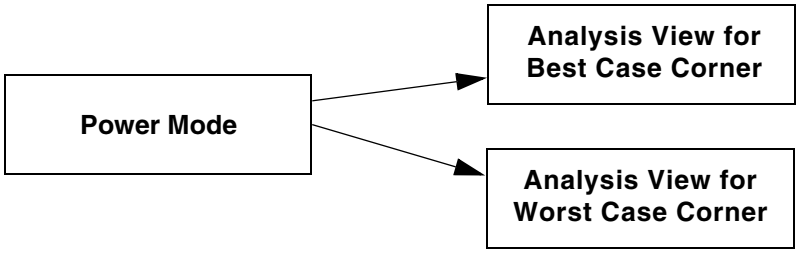


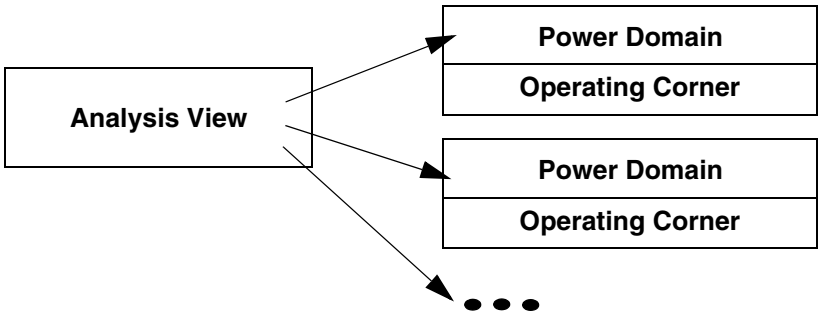
Figure 2-1 shows the typical operating voltage for each power domain. To check if the design functions correctly when slightly different operating conditions apply, typically a multi-corner timing analysis is done for the worst and best case corner. A pure MSV design has only one state of the design, so two views can be considered as shown in Figure 2-2. The design must function correctly in both corners for the same set of timing constraints that is typically specified for a given state of the design (power mode).

Figure 2-2 Multi-Corner Timing Analysis



When analyzing a view of the design, you need to indicate the corners at which the power domains are operating, as shown in Figure 2-3.

Figure 2-3 Relation of Analysis View with Power Domains and Operating Corners



Operating corners are not only characterized by a set of operating conditions, but also by the library sets to be used for that corner, because the timing and power characteristics of the cells depend on the operating conditions. Typically different libraries are used for the different corners.

Table 2-1 on page 19 indicates for the design of Figure 2-1 on page 18 the libraries that must be used to analyze the views.

Table 2-1 Operating Corners for the Power Domains of the MSV Design

Operating Corner	Operating Condition			Libraries
Name	process	temperature	voltage	
BC_PD1	1	0	0.88	lib1_bc, lib2_bc
BC_PD2	1	0	1.1	lib3_bc
BC_PD3	1	0	1.32	lib4_bc
WC_PD1	1	125	0.72	lib1_wc, lib2_wc
WC_PD2	1	125	0.9	lib3_wc
WC_PD3	1	125	1.08	lib4_wc

As it is obvious that all this information is related to the power intent of the design, it makes sense to describe this information in the CPF file.

Complete CPF File for MSV Example

```
set_cpf_version 1.1
################################################
#       Technology part of the CPF
################################################
# define the library sets
define_library_set -name set1_bc -libraries {lib1_bc lib2_bc}
define_library_set -name set1_wc -libraries {lib1_wc lib2_wc}
define_library_set -name set2_bc -libraries lib3_bc
define_library_set -name set2_wc -libraries lib3_wc
define_library_set -name set3_bc -libraries lib4_bc
define_library_set -name set3_wc -libraries lib4_wc
# define the level shifters
define_level_shifter_cell -cells LVLLHEHX* \
    -input_voltage_range 0.8 \
    -output_voltage_range 1.0 \
    -output_power_pin VDD \
    -ground VSS \
    -direction up \
    -valid_location from
define_level_shifter_cell -cells LVLLHX* \
    -input_voltage_range 0.8 \
    -output_voltage_range 1.2 \
    -output_power_pin VDD \
    -direction up \
    -ground VSS \
    -valid_location to
define_level_shifter_cell -cells LVLLHELX* \
    -input_voltage_range 1.0 \
    -output_voltage_range 1.2 \
    -output_power_pin VDD \
    -direction up \
    -ground VSS \
    -valid_location to
################################################
#       Design part of the CPF
################################################
set_design top
# create power domains
create_power_domain -name PD1 -default
create_power_domain -name PD2 -instances instance_B
create_power_domain -name PD3 -instances instance_C
# create nominal conditions
create_nominal_condition -name low -voltage 0.8
create_nominal_condition -name medium -voltage 1.0
create_nominal_condition -name high -voltage 1.2
```

```
# create power mode
create_power_mode -name PM -domain_conditions {PD1@low PD2@medium PD3@high} \
-default
# associate library sets with nominal conditions
update_nominal_condition -name low -library_set set1_wc
update_nominal_condition -name medium -library_set set2_wc
update_nominal_condition -name high -library_set set3_wc
# create rules for level shifter insertion
create_level_shifter_rule -name lsr1 -from PD1 -to PD2
create_level_shifter_rule -name lsr2 -from PD2 -to PD3
create_level_shifter_rule -name lsr3 -from PD1 -to PD3
####################################################
# Additional Information for Logic Synthesis
####################################################
# specify power targets
set_power_target -leakage 30 -dynamic 250
# specify timing constraints
update_power_mode -name PM -sdc_files top.sdc
# specify activity information
update_power_mode -name PM -activity_file top.tcf -activity_file_weight 100
# update the rules
update_level_shifter_rules -names lsr1 -location from
update_level_shifter_rules -names {lsr2 lsr3} -location to
####################################################
# Additional Information for Physical Implementation
####################################################
# declare power and ground nets
create_ground_nets -nets VSS
create_power_nets -nets VDD1 -voltage 0.8
create_power_nets -nets VDD2 -voltage 1.0
create_power_nets -nets VDD3 -voltage 1.2
# (optional) create global connections
# create_global_connection -net VSS -pins VSS
# create_global_connection -domain PD1 -net VDD1 -pins VDD
# create_global_connection -domain PD2 -net VDD2 -pins VDD
# create_global_connection -domain PD3 -net VDD3 -pins VDD
# add implementation info for power domains
update_power_domain -name PD1 -primary_power_net VDD1 -primary_ground_net VSS
update_power_domain -name PD2 -primary_power_net VDD2 -primary_ground_net VSS
update_power_domain -name PD3 -primary_power_net VDD3 -primary_ground_net VSS
# create operating corners
create_operating_corner -name BC_PD1 \
    -process 1 -temperature 0 -voltage 0.88 -library_set set1_bc
```

```
create_operating_corner -name BC_PD2 \
    -process 1 -temperature 0 -voltage 1.0 -library_set set2_bc
create_operating_corner -name BC_PD3 \
    -process 1 -temperature 0 -voltage 1.32 -library_set set3_bc
create_operating_corner -name WC_PD1 \
    -process 1 -temperature 125 -voltage 0.72 -library_set set1_wc
create_operating_corner -name WC_PD2 \
    -process 1 -temperature 125 -voltage 0.9 -library_set set2_wc
create_operating_corner -name WC_PD3 \
    -process 1 -temperature 125 -voltage 1.08 -library_set set3_wc
# create analysis views
create_analysis_view -name AV_BC -mode PM \
    -domain_corners {PD1@BC_PD1 PD2@BC_PD2 PD3@BC_PD3}
create_analysis_view -name AV_WC -mode PM \
    -domain_corners {PD1@WC_PD1 PD2@WC_PD2 PD3@WC_PD3}
end_design
```

Steps to Create the CPF File for MSV Design

This section describes the information to include in a CPF file for an MSV design. The example shown in Figure 2-1 on page 18 is used throughout this section.

A CPF file has technology-related information and design-related information.

For an MSV design, the technology-related information lists the libraries that you want to use for the design and identifies the library cells that can be used as level shifters.

- Specifying the Libraries on page 24
- Specifying the Level Shifter Cells to Use on page 25

The design-related information captures the power intent and constraints.

- Declaring the Design Described in the CPF File on page 84
- Specifying Units Used on page 26
- Specifying the Naming Styles Used on page 27
- Specifying the Power Domains on page 28
- Specifying the Operating Voltages Used in the Design on page 28
- Associating a Nominal Condition with a Power Domain on page 29
- Specifying the Libraries to Use for a Condition on page 29
- Specifying the Rules to Create Level Shifter Logic on page 30
- Specifying Power Constraints on page 31
- Specifying Timing Constraints on page 31
- Specifying Activity Information on page 31
- Updating the Rules with Implementation Information on page 32

The following information is needed for physical implementation:

- Specifying the Global Power and Ground Nets on page 33
- Specifying the Global Connections on page 34
- Specifying Additional Information for Power and Ground Routing on page 34
- Specifying the Operating Corners on page 35
- Specifying the Analysis Views on page 36

Specifying the Libraries

➤ To group libraries that are characterized for a specific set of operating conditions, use the define_library_set command:

```
define_library_set -name library_set
    -libraries library_list
```

For the example in Figure 2-1 on page 18, six library sets are defined: one library set for the best and worst case operating conditions of each power domain.

```
define_library_set -name set1_bc -libraries {lib1_bc lib2_bc}
define_library_set -name set1_wc -libraries {lib1_wc lib2_wc}
define_library_set -name set2_bc -libraries lib3_bc
define_library_set -name set2_wc -libraries lib3_wc
define_library_set -name set3_bc -libraries lib4_bc
define_library_set -name set2_wc -libraries lib3_wc
```

Specifying the Level Shifter Cells to Use

➤ To identify which cells in the libraries can be used as level shifters, use the `define_level_shifter_cell` command.

For more information on how to model different types of level shifters, refer to Modeling Level Shifters on page 150.

For applications that do not read .lib files, you must specify the library cells to allow the application to identify the instances of these cells in the netlist.

An MSV design may need different level shifters depending on the voltage gaps to bridge. For the example in Figure 2-1 on page 18, the following command defines a set of level shifters that can be used from a power domain operating on 0.8V to a power domain operating on1.0V.

```
define_level_shifter_cell -cells LVLLHEHX* \
    -input_voltage_range 0.8 \
    -output_power_pin VDD \
    -output_voltage_range 1.0 \
    -direction up \
    -ground VSS \
    -valid_location from
```

Declaring the Design Described in the CPF File

➤ To identify the design for which the CPF file is created, use the following command:

```
set_design module
```

where *module* refers to the name of the top module of the design to which the power information in the CPF file applies.

For the example in Figure 2-1 on page 18:

```
set_design top
```

➤ To indicate when the power information for this module ends, use the following command:

```
end_design
```

Specifying Units Used

You can specify the units that will be used for the power and time values in CPF commands.

➤ To specify the power unit used, use the following command

```
set_power_unit [pW|nW|uW|mW|W]
```

Default: mW

➤ To specify the time unit used, use the following command

```
set_time_unit [ns|us|ms]
```

Default: `ns`

Note: These commands are optional if you use the default values.

Specifying the Naming Styles Used

➤ To specify the hierarchy separator used in the CPF file, use the following command

`set_hierarchy_separator [character]`

The default separator is the period (.).

The format of a name in RTL and in the netlist can be different. When you want to use the RTL names in the CPF file, but you are reading a gate-level netlist, you need to specify how the base name and bit information are represented in the netlist.

➤ To specify the format used to name flip-flops and latches in the netlist starting from the register names in the RTL description, use the following command

`set_register_naming_style [string%s]`

Default: `_reg%s`

➤ To specify the format used to name the design objects in the netlist starting from multi-bit arrays in the RTL description, use the following command

`set_array_naming_style [string]`

Default: `\[%d\]`

For more information on these two commands, refer to Individual Registers Names in the *Common Power Format Language Reference.*

Note: These three commands are optional if you use the default values.

Specifying the Power Domains

➤ To identify portions of the design that operate on the same voltage, associate these portions with a power domain using the create_power_domain command:

```
create_power_domain -name power_domain
    [-instances instance_list] [-boundary_ports pin_list] [-default]
```

Note: The following options of create_power_domain are irrelevant for a pure MSV (non-switchable) design: -shutoff_condition, external_controlled_shutoff, -default_isolation_condition, -default_restore_edge, -default_save_edge, -default_restore_level, -default_save_level, -power_up_states, -active_state_conditions, and -base_domains.

For the example in Figure 2-1 on page 18:

```
create-power_domain -name PD1 -default
create_power_domain -name PD2 -instances inst_B
create_power_domain -name PD3 -instances inst_C
```

Note: CPF requires that the top module belongs to the default power domain.

Specifying the Operating Voltages Used in the Design

In CPF, operating voltages are associated with *nominal conditions*.

➤ To specify the operating voltages used in the design, use the create_nominal_condition command:

```
create_nominal_condition -name string
-voltage {voltage | voltage_list}
[-ground_voltage {voltage | voltage_list}]
```

Note: The following options of create_nominal_condition are irrelevant for a pure MSV design: -state, -pmos_bias_voltage and -nmos_bias_voltage.

For the example in Figure 2-1 on page 18 three nominal conditions are defined.

```
create_nominal_condition -name low -voltage 0.8
create_nominal_condition -name medium -voltage 1.0
create_nominal_condition -name high -voltage 1.2
```

Tip

In CPF, operating voltages are not directly associated with power domains. When using other low power techniques such as power shut off (PSO) or dynamic voltage frequency scaling (DVFS), the operating voltage of a power domain can change and different power domains can use the same operating voltage at different times.

Associating a Nominal Condition with a Power Domain

In CPF, nominal conditions are associated with power domains for a given design mode (here referred to as *power mode*).

A pure MSV design (a design that uses multiple supply voltages but no other low power techniques such as PSO or DVFS methodology) is considered to have only one power mode and each power domain can be associated with only one nominal condition.

➤ To associate the nominal conditions with the power domains, use the `create_power_mode` command:

```
create_power_mode -name string
    -domain_conditions domain_condition_list -default
```

Note: The -`group_modes` option of the `create_power_mode` command is only relevant for a hierarchical flow. For more information, refer to Chapter 4, “Hierarchical Flow.”

Use the following format to specify a *domain condition* (association of a power domain with its nominal condition):

```
domain_name@nominal_condition_name
```

For the example in Figure 2-1 on page 18:

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

Note: CPF requires that one power mode is specified as the default power mode.

Specifying the Libraries to Use for a Condition

You already grouped libraries that are characterized for a specific set of operating conditions in a library set.

➤ To specify which library set to use for a specific nominal condition, use the `update_nominal_condition` command:

```
update_nominal_condition -name condition
    -library_set library_set
```

Typically, you specify here the libraries for the worst case condition. For the example in Figure 2-1 on page 18:

```
update_nominal_condition -name low -library_set set1_wc
update_nominal_condition -name medium -library_set set2_wc
update_nominal_condition -name high -library_set set3_wc
```

Specifying the Rules to Create Level Shifter Logic

Depending on your technology, you may need level shifters when passing any signals

- From a power domain with a lower voltage to a power domain with a higher voltage
- From a power domain with a higher voltage to a power domain with a lower voltage
- In both cases

➤ To create the rule to be used between power domains or a set of pins, use the create_level_shifter_rule command:

```
create_level_shifter_rule -name string
    {-pins pin_list | -from power_domain_list | -to power_domain_list}...
    [-exclude pin_list]
```

For the example in Figure 2-1 on page 18:

```
create_level_shifter_rule -name lsr1 -from PD1 -to PD2
create_level_shifter_rule -name lsr2 -from PD2 -to PD3
create_level_shifter_rule -name lsr3 -from PD1 -to PD3
```

Specifying Power Constraints

Note: This information is optional in the CPF file.

➤ To specify the targets for leakage and dynamic power in the current design, use the set_power_target command:

```
set_power_target
    { -leakage float | -dynamic float
    | -leakage float -dynamic float }
```

The power constraints must be specified using the power units defined with the set_power_unit command.

For the example in Figure 2-1 on page 18:

```
set_power_target -leakage 30 -dynamic 250
```

Specifying Timing Constraints

Note: This information is optional in the CPF file.

➤ To specify the timing constraints for the current design, use the -sdc_files option of the update_power_mode command:

```
update_power_mode -name mode
    {-sdc_files | -setup_sdc_files | -hold_sdc_files} sdc_file_list
```

For the example in Figure 2-1 on page 18:

```
update_power_mode -name PM -sdc_files top.sdc
```

Specifying Activity Information

Note: This information is optional in the CPF file.

➤ To specify the activity information that can be used for power analysis, use

```
update_power_mode -name mode
    -activity_file file -activity_file_weight weight
```

Supported formats for the activity files are VCD, TCF, and SAIF.

If the design has several modes, you can specify the relative weight of the activities per mode for optimization purpose. Because an MSV design has only one power mode, the weight for the file must be 100.

For the example in Figure 2-1 on page 18:

```
update_power_mode -name PM -activity_file top.tcf -activity_file_weight 100
```

Updating the Rules with Implementation Information

You can specify additional information for level shifters, such as, where to place them, what cells to use, or what prefix to use for the added level shifters in the design.

➤ To specify the location or the type of level shifters to be used, use the update_level_shifter_rules command:

```
update_level_shifter_rules -names rule_list
    { -location {from | to | parent | any}
    | -within_hierarchy instance
    | -cells cell_list
    | -prefix string }...
```

For the example in Figure 2-1 on page 18:

```
update_level_shifter_rules -names lsr1 -location from
update_level_shifter_rules -names {lsr2 lsr3} -location to
```

Specifying the Global Power and Ground Nets

➤ To declare (or create) the nets connected to the power supplies, use the create_ground_nets and commands:

```
create_ground_nets -nets net_list
    [-voltage string]
    [-user_attributes string_list]
    [-peak_ir_drop_limit float]
    [-average_ir_drop_limit float]

create_power_nets -nets net_list
    [-voltage string]
    [-user_attributes string_list]
    [-peak_ir_drop_limit float]
    [-average_ir_drop_limit float]
```

Note: The following options of `create_ground_nets` and `create_power_nets` are irrelevant for the pure MSV design: `-external_shutoff_condition`, and `-internal.`

For the example in Figure 2-1 on page 18:

```
create_ground_nets -nets VSS
create_power_nets -nets VDD1 -voltage 0.8
create_power_nets -nets VDD2 -voltage 1.0
create_power_nets -nets VDD3 -voltage 1.2
```

Specifying the Global Connections

➤ To specify how to connect global nets, such as power and ground nets, use the create_global_connection command:

```
create_global_connection
    -net net
    -pins pin_list
    [-domain domain | -instances instance_list]
```

For the example in Figure 2-1 on page 18:

```
create_global_connection -net VSS -pins VSS
create_global_connection -domain PD1 -net VDD1 -pins VDD
create_global_connection -domain PD2 -net VDD2 -pins VDD
create_global_connection -domain PD3 -net VDD3 -pins VDD
```

Specifying Additional Information for Power and Ground Routing

➤ To specify additional information that applies to power and ground routing, use

```
update_power_domain
    -name domain
    { -primary_power_net net | -primary_ground_net net } ...
```

For the example in Figure 2-1 on page 18:

```
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 VDD3
```

Here the `update_power_domain` command for power domain `PD1` indicates that `VDD1` is the main power net for all cells of power domain `PD1`.

Specifying the Operating Corners

The design must be able to perform under different sets of operating conditions (process, voltage, and temperature values). Because the timing and power characteristics of the cells depend on the operating conditions, different library sets that contain the characterization information for the different conditions are used. An operating corner shows which library set to use for a given operating condition.

Tip

Different portions of the design can operate at the same voltage and yet use dedicated library sets. In this case, it is possible to have multiple operating corners with the same operating conditions.

➤ To define an operating corner, use the `create_operating_corner` command.

```
create_operating_corner
    -name string
    -voltage float [-ground_voltage float]
    [-process float]
    [-temperature float]
    -library_set library_set
```

Note: The `-pmos_bias_voltage` and `-nmos_bias_voltage` options of `create_operating_corner` are irrelevant for a design not using substrate biasing.

For the design in Figure 2-1 on page 18, six operating corners were specified.

The following command specifies to use library set `set1_bc` for operating corner `BC_PD1`. The process value of the operating corner is 1, the temperature is 0°C and the operating voltage is 0.88V. This operating corner is defined for best case conditions.

```
create_operating_corner -name BC_PD1 \
    -process 1 -temperature 0 -voltage 0.88 -library_set set1_bc
```

Specifying the Analysis Views

The design must function correctly in each power mode not only under typical conditions, but also under extreme conditions. Typically a multi-mode multi-corner timing analysis will be done for the worst case and the best case conditions.

An analysis view associates a specific operating corner with each power domain in the specified power mode. You need to make sure that all operating corners for a view correspond to either best or worst case conditions.

➤ To define an analysis view, use the `create_analysis_view` command.

```
create_analysis_view
    -name string
    -mode mode
    -domain_corners domain_corner_list
    [-user_attributes string_list]
```

Note: The `-group_views` option of `create_analysis_view` is only relevant in a hierarchical flow. For more information, refer to Chapter 4, “Hierarchical Flow.”

Use the following format to specify a domain corner:

domain_name@corner_name

Tip

The CPF file can contain several analysis views with the same domain and corner information for a power mode. For a multi-mode multi-corner analysis, some implementation and timing analysis tools need unique views to associate with different parasitic corners.

For the design in Figure 2-1 on page 18, two analysis views were specified.

The following command specifies analysis view `AV_BC` for power mode `PM1`. For this view the operating corners for the best case operating conditions are associated with the three power domains.

```
create_analysis_view -name AV_BC -mode PM1 \
    -domain_corners {PD1@BC_PD1 PD2@BC_PD2 PD3@BC_PD3}
```

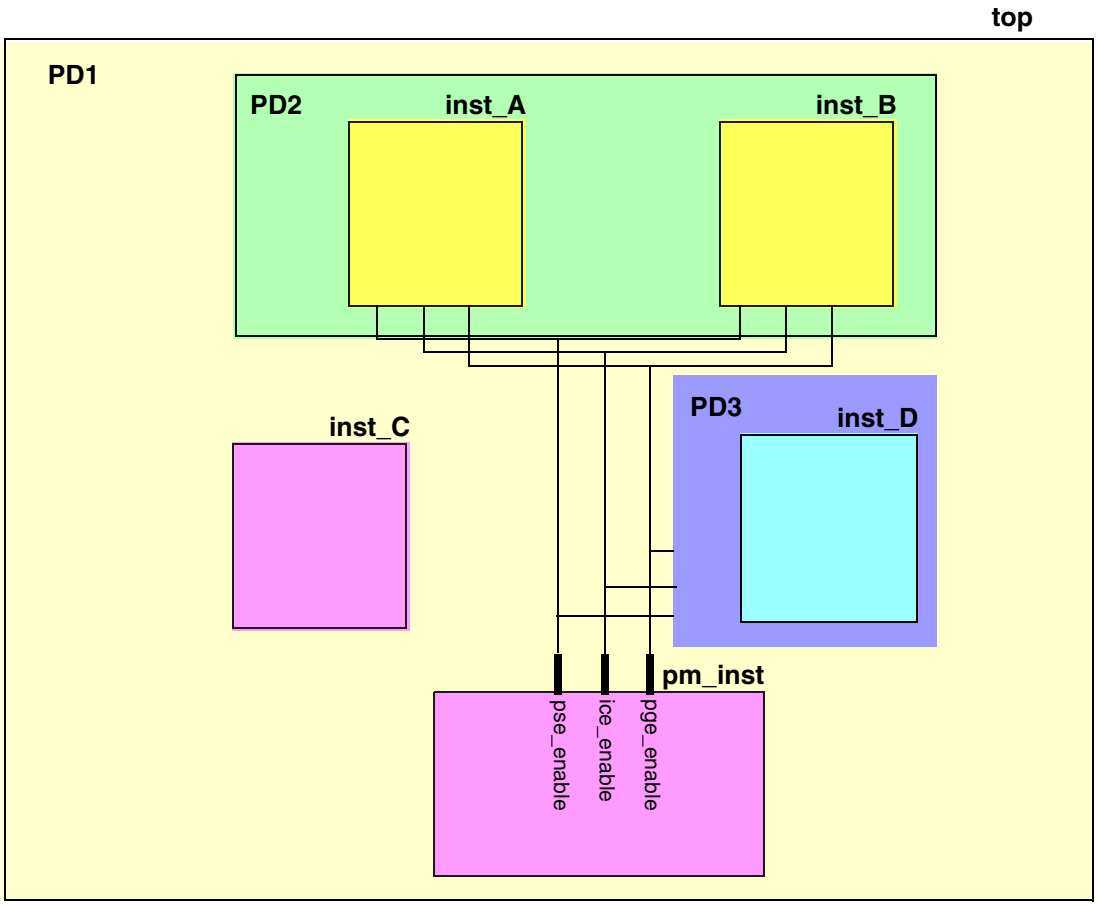
Creating a CPF File for a Design Using PSO Methodology

A design using power shut off (**PSO**) implementation is a design of which some portions can be switched on and off as needed (or possibly) to save leakage and dynamic power.

Logic blocks (hierarchical instances), leaf instances, and pins that use the same *main* power supply and that can be simultaneously switched on or off are said to belong to the same ***power domain***. The example design in Figure 2-4 has three power domains:

- The top-level of the design, `top`, and hiearchical instances, `inst_C` and `pm_inst`, are always switched on: they belong to domain `PD1`
- Hierarchical instances `inst_A` and `inst_B` are always switched on and off simultaneously: they belong to power domain `PD2`
- Hierarchical instance `inst_D` can be switched on and off independently from hierarchical instances `inst_A` and `inst_B`: it belongs to power domain `PD3`

Figure 2-4 Example of Design with PSO



Power domain `PD1` is never powered down. It is called an unswitched domain.

Power domains `PD2` and `PD3` can be powered down. They are referred to as switchable domains. CPF distinguishes between internal switchable, and on-chip controlled external switchable domains.

A steady state of the design in which some power domains are switched on and some power domains are switched off is called a ***power mode***. In a power mode, each power domain operates on a specific nominal condition. Different timing constraints can be associated with each power mode. Table 2-2 shows the three power modes of the example design.

Table 2-2 Power Modes

Power Mode	Power Domain		
	PD1	PD2	PD3
PM1	1.1V	1.1V	1.1V
PM2	1.1V	0.0V	1.1V
PM3	1.1V	0.0V	0.0V

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

To prevent that unknown states in the power domains that are powered down propagate to the domains that remain powered on ***isolation cells*** are needed at the boundaries of the power domains that are powered down. Most of the time, isolation cells are inserted at the output boundaries of the powered down domains. You can, however, also insert isolation cells at the input boundaries.

To facilitate powered down blocks to resume normal operation, ***state retention cells*** can be used for some sequential cells to keep their previous state prior to power down.

For switchable domains you need to indicate how the power supply is connected and disconnected from the gates.

- For internal switchable domains, you must add ***power switch logic.***
- For external switchable domains, the power switch logic is not part of the chip, so you must indicate that an *external power shut-off method* is used.

For this example we are assuming that power domains `PD2` and `PD3` are internal switchable.

Special control signals are used to shut down a power domain, enable state retention, and control the working of the power switch logic. Table 2-3 on page 39 shows the signals used in this example.

Table 2-3 Signals Controlling the Power Domains

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

When a domain is switchable, it derives its power from another power domain through either internal or external power switch logic.

In this example, power domain `PD2` derives its power from power domain `PD1`, then `PD1` is called the ***secondary*** (or base) domain for `PD2`, which is referred to as the ***primary*** (or derived) domain.

When defining a (primary) power domain you can indicate its secondary domain and under which condition the domain will be shut down.

The majority of instances in a power domain are driven by the same power supply. For switchable domains, it is the primary power and ground nets of the (primary) power domain to which the instances belong that provide the power supply to the power and ground pins (follow-pins) of the cells.

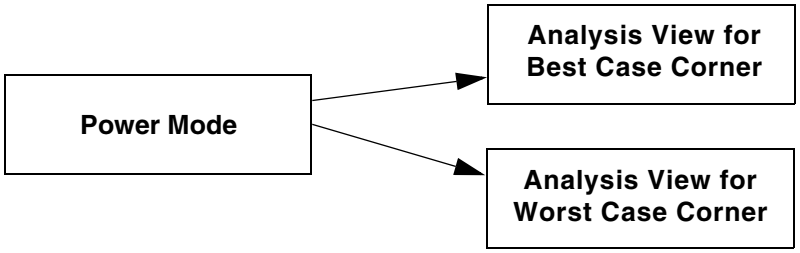
On the other hand, isolation cells and state retention cells are driven by multiple power supplies. These special low power instances have at least two sets of power and ground pins, and are therefore associated with two power domains: the ***primary*** domain is the domain that provides the power supply to their primary set of power and ground pins. The ***secondary*** domain is the domain whose primary power and ground nets provide the power supply to the secondary power and ground pins of the special low power instances.

It is recommended that you specify the secondary domain for these special low power instances, but if you do not specify this information, the tools can use some rules to derive the secondary domain. For more information, refer to Secondary Power Domain of Isolation Instances, and Secondary Domain of Retention Logic.

If the design can operate in different power modes, you need to check if the design functions correctly in each of these modes not only at the typical conditions but also when slightly

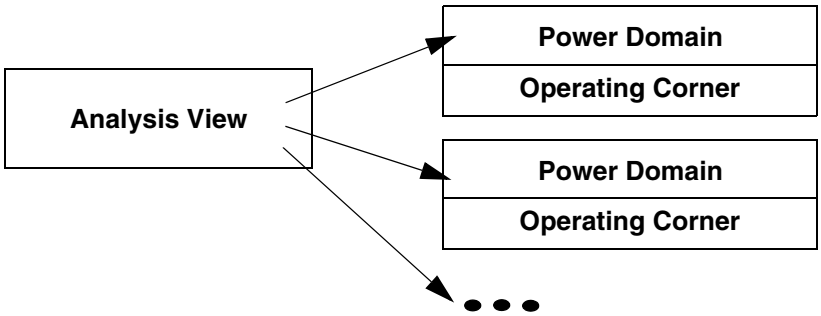
different operating conditions apply. Typically a multi-mode multi-corner timing analysis will be done for the worst case and the best case corner. The design must function correctly in both corners for the same set of timing constraints that is specified for that power mode. Figure 2-5 on page 40 shows that an analysis (view) can be done for each corner of the power mode.

Figure 2-5 Multi-Corner Timing Analysis for a Power Mode



When analyzing a view of the design, you need to indicate the corners at which the power domains are operating, as shown in Figure 2-6.

Figure 2-6 Relation of Analysis View with Power Domains and Operating Corners



Operating corners are not only characterized by a set of operating conditions, but also by the library sets to be used for that corner, because the timing and power characteristics of the cells depend on the operating conditions. Typically different libraries are used for the different corners.

Table 2-4 indicates which library sets must be used for the design in Figure 2-4 on page 37 to check the conditions.

Table 2-4 Operating Corners

Operating Corner	Operating Condition			Library set
Name	process	temperature	voltage	
BC	1	0	0.99	lib1_bc, lib2_bc
WC	1	125	1.21	lib1_wc, lib2_wc

As is obvious that all this information is related to the power intent of the design, it makes sense to describe this information in the CPF file.

Complete CPF File for PSO Example

```
set_cpf_version 1.1
################################################
#       Technology part of the CPF
################################################
# define the library sets
define_library_set -name set1_wc -libraries {lib1_wc lib2_wc}
define_library_set -name set1_bc -libraries {lib1_bc lib2_bc}
# define the isolation cells
define_isolation_cell -cells ISOLN* -enable EN -valid_location on
# define the always on cell
define_always_on_cell -cells "BUFGX2M BUFGX8M INVGX2M INVGX8M"
# define the state retention cell
define_state_retention_cell -cells *DRFF* -restore_function RETN
# define the power switch cells
define_power_switch_cell -cells "hd8DM hd16DM hd32DM hd64DM"  \
    -stage_1_enable SLEEP -type header
define_power_switch_cell -cells "hd8M hd16M hd32M hd64M"  \
    -stage_1_enable !SLEEP -type header
define_power_switch_cell -cells "ft8DM ft16DM" \
    -stage_1_enable !SLEEPN -type footer
define_power_switch_cell -cells "ft8M ft16M"  \
    -stage_1_enable SLEEPN -type footer
################################################
#       Design part of the CPF
################################################
# identify the design for which the CPF file is created
set_design top
# create power domains
create_power_domain -name PD1 -default
create_power_domain -name PD2 -instances {inst_A inst_B} \
-shutoff_condition {pse_enable[0]} -base_domains PD1
create_power_domain -name PD3 -instances inst_D \
-shutoff_condition {pse_enable[1]} -base_domains PD1
# create nominal conditions
create_nominal_condition -name off -voltage 0
create_nominal_condition -name on -voltage 1.1
# create power modes
create_power_mode -name PM1 -domain_conditions {PD1@on PD2@on PD3@on} -default
create_power_mode -name PM2 -domain_conditions {PD1@on PD3@on}
create_power_mode -name PM3 -domain_conditions {PD1@on}
# associate library sets with nominal conditions
update_nominal_condition -name on -library_set set1_wc
```

```
# create rules for isolation logic insertion
create_isolation_rule -name iso1 -from PD2 \
-isolation_condition {pm_inst.ice_enable[0]}
create_isolation_rule -name iso2 -to PD1\
-isolation_condition {pm_inst.ice_enable[1]} -isolation_output high
# create rules for state retention insertion
create_state_retention_rule -name st1 -domain PD2 \
-restore_edge {!pm_inst.pge_enable[0]}
create_state_retention_rule -name st2 -domain PD3 \
-restore_edge {!pm_inst.pge_enable[1]}
#####################################################
# Additional Information for Logic Synthesis
#####################################################
# specify power targets
set_power_target -leakage 30 -dynamic 250
# specify timing constraints
update_power_mode -name PM1 -sdc_files pm1.sdc
update_power_mode -name PM2 -sdc_files pm2.sdc
update_power_mode -name PM1 -activity_file top.tcf -activity_file_weight 100
# update the rules with implementation info
update_isolation_rules -names iso1 -location to -cells ISOLNX2M
update_isolation_rules -names iso2 -location to -cells ISOLNX2M
#####################################################
# Additional Information for Physical Implementation
#####################################################
# declare power and ground nets
create_power_nets -nets VDD -voltage 1.1
create_power_nets -nets {VDD_SW1 VDD_SW2} -internal
create_ground_nets -nets VSS -voltage 0
# (optional) create global connections
create_global_connection -net VDD -pins VDD
create_global_connection -net VSS -pins VSS
# rules for power switch insertion
create_power_switch_rule -name SW1 -domain PD2 -external_power_net VDD
create_power_switch_rule -name SW2 -domain PD3 -external_power_net VDD
update_power_switch_rule -name SW1 -cells hd32M -prefix CDN_
update_power_switch_rule -name SW2 -cells hd32M -prefix CDN_
# add implementation info for power domains
update_power_domain -name PD1 -primary_power_net VDD -primary_ground_net VSS
update_power_domain -name PD2 -primary_power_net VDD_SW1 -primary_ground_net VSS
update_power_domain -name PD3 -primary_power_net VDD_SW2 -primary_ground_net VSS
# create operating corners
create_operating_corner -name BC \
    -process 1 -temperature 0 -voltage 1.21 -library_set set1_bc
```

```
create_operating_corner -name WC \
    -process 1 -temperature 125 -voltage 0.99 -library_set set1_wc
# create analysis views
create_analysis_view -name AV_PM1_bc -mode PM1 \
    -domain_corners {PD1@BC PD2@BC PD3@BC}
create_analysis_view -name AV_PM1_wv -mode PM1 \
    -domain_corners {PD1@WC PD2@WC PD3@WC}
create_analysis_view -name AV_PM2_bc -mode PM2 \
    -domain_corners {PD1@BC PD2@BC PD3@BC}
create_analysis_view -name AV_PM2_wc -mode PM2 \
    -domain_corners {PD1@WC PD2@WC PD3@WC}
create_analysis_view -name AV_PM3_bc -mode PM3 \
    -domain_corners {PD1@BC PD2@BC PD3@BC}
create_analysis_view -name AV_PM3_wc -mode PM3 \
    -domain_corners {PD1@WC PD2@WC PD3@WC}
# indicate when the power information for the design ends
end_design
```

Steps to Create the CPF File for Design Using PSO

This section describes the information to include in a CPF file for a design using the PSO methodology. The example shown in Figure 2-4 on page 38 is used throughout this section.

A CPF file has technology-related information and design-related information.

For a design using the PSO methodology, the technology-related information lists the libraries that you want to use for the design and identifies the library cells that can be used as isolation cells, power switch cells and state retention cells.

- Specifying the Libraries on page 47
- Specifying the Isolation Cells to Use on page 47
- Specifying the Always-On Cells on page 48
- Specifying the State Retention Cells to be Used on page 49
- Specifying the Power Switch Cells to be Used on page 50

The design-related information captures the power intent and constraints.

- Declaring the Design Described in the CPF File on page 51
- Specifying Units Used on page 51
- Specifying Naming Styles Used on page 52
- Specifying the Power Domains on page 53
- Specifying the Operating Voltage Used in the Design on page 53
- Specifying the Static Behavior in each Power Mode on page 54
- Specifying the Libraries to Use for a Condition on page 54
- Specifying the Rules to Create Isolation Logic on page 55
- Specifying the Rules to Create State Retention Logic on page 56
- Specifying Power Constraints on page 56
- Specifying Timing Constraints on page 57
- Specifying Activity Information on page 57
- Updating the Rules with Information for Implementation on page 58

The following information is needed for physical implementation:

Specifying the Libraries

➤ To group libraries that are characterized for a specific set of operating conditions, use the define_library_set command:

```
define_library_set -name library_set
    -libraries library_list
```

For the example in Figure 2-4 on page 37, we assume only one main power supply, which implies that the definition of one library set is sufficient.

```
define_library_set -name set1_wc -libraries {lib1_wc lib2_wc}

define_library_set -name set1_bc -libraries {lib1_bc lib2_bc}
```

Specifying the Isolation Cells to Use

➤ To identify which cells in the libraries can be used as isolation cells, use the define_isolation_cell command.

For more information on how to model different types of isolation cells, refer to Modeling Isolation Cells on page 165.

For applications that do not read .lib files, you must specify these library cells to allow these applications to identify instances of isolation cells in the netlist.

For the example in Figure 2-4 on page 37:

```
define_isolation_cell -cells ISOLN* -enable EN -valid_location on
```

Specifying the Always-On Cells

Always-on cells are special cells whose power supply has to be continuous on even when the power supply for the rest of the logic in the power domain is off.

Always-on cells are used for example

- To drive the control signals of the state retention cells in a domain that is being powered down
- In combination with isolation cells that are inserted in the power domain that is switched off to ensure that the driver of the enable pin of the isolation cells is never switched off.

➤ To identify which cells in the libraries can be used as always-on cells, use the define_always_on_cell command.

```
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 ]
```

Note: Outputs of cells that are always on, are always-on drivers.

For applications that do not read .lib files, you must specify these library cells to allow these applications to identify the instances of these cells in the netlist.

For the example in Figure 2-4 on page 37,

```
define_always_on_cell -cells "BUFGX2M BUFGX8M INVGX2M INVGX8M"
```

Specifying the State Retention Cells to be Used

➤ To identify which cells in the libraries can be used as state retention cells, use the `define_state_retention_cell` command.

For more information on how to model different types of state retention cells, refer to Modeling State Retention Cells on page 181.

For applications that do not read .lib files, you must specify these library cells to allow these applications to identify the instances of these cells in the netlist.

For the example in Figure 2-4 on page 37,

```
define_state_retention_cell -cells *DRFF* -restore_function RETN
```

This command indicates that when the restore pin `RETN` is set to 1, the state of the specified cells will be restored to the value saved after exiting power shut-off mode.

Specifying the Power Switch Cells to be Used

➤ To identify which cells in the libraries can be used as power switch cells, use the `define_power_switch_cell` command.

For more information on how to model different types of power switch cells, refer to Modeling Power Switch Cells on page 188.

For applications that do not read .lib files, you must specify the library cells to allow the applications to identify the instances of these cells in the netlist.

For the example in Figure 2-4 on page 37,

```
define_power_switch_cell -cells "hd8DM hd16DM hd32DM hd64DM"  \
    -stage_1_enable SLEEP -type header
define_power_switch_cell -cells "hd8M hd16M hd32M hd64M"  \
    -stage_1_enable !SLEEP -type header
define_power_switch_cell -cells "ft8DM ft16DM" \
    -stage_1_enable !SLEEPN -type footer
define_power_switch_cell -cells "ft8M ft16M"  \
    -stage_1_enable SLEEPN -type footer
```

Declaring the Design Described in the CPF File

➤ To identify the design for which the CPF file is created, use the following command:

```
set_design module
```

where *module* refers to the name of the top module of the design to which the power information in the CPF file applies.

For the example in Figure 2-4 on page 38:

```
set_design top
```

➤ To indicate when the power information for this module ends, use the following command:

```
end_design
```

Specifying Units Used

You can specify the units that will be used for the power and time values in CPF commands.

➤ To specify the power unit used, use the following command

```
set_power_unit [pW|nW|uW|mW|W]
```

Default: mW

➤ To specify the time unit used, use the following command

```
set_time_unit [ns|us|ms]
```

Default: `ns`

Note: These commands are optional if you use the default values.

Specifying Naming Styles Used

➤ To specify the hierarchy separator used in the CPF file, use the following command

`set_hierarchy_separator [character]`

The default separator is the period (.).

The format of a name in RTL and in the netlist can be different. When you want to use the RTL names in the CPF file, but you are reading a gate-level netlist, you need to specify how the base name and bit information are represented in the netlist.

➤ To specify the format used to name flip-flops and latches in the netlist starting from the register names in the RTL description, use the following command

`set_register_naming_style [string%s]`

Default: `_reg%s`

➤ To specify the format used to name the design objects in the netlist starting from multi-bit arrays in the RTL description, use the following command

`set_array_naming_style [string]`

Default: `\[%d\]`

For more information on these two commands, refer to Individual Registers Names in the *Common Power Format Language Reference.*

Note: These three commands are optional if you use the default values.

Specifying the Power Domains

➤ To identify portions of the design that operate on the same voltage and that can be simultaneously switched on or off, associate these portions with a power domain using the create_power_domain command:

```
create_power_domain -name power_domain
    [-instances instance_list] [-boundary_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} ]
    [ -active_state_conditions active_state_condition_list ]
    [ -base_domains domain_list]
```

The -shutoff_condition determines when the power domain is switched off. If this option is not specified, the power domain is an unswitched domain.

For the example in Figure 2-4 on page 37:

```
create-power_domain -name PD1 -default

create_power_domain -name PD2 -instances {inst_A inst_B} \
-shutoff_condition {pse_enable[0]} -base_domains PD1

create_power_domain -name PD3 -instances inst_D \
-shutoff_condition {pm_inst.pse_enable[1]} -base_domains PD1
```

Power domains PD2 and PD3 have power domain PD1 as their secondary power domain.

Note: CPF requires that the top module belongs to the default power domain.

Specifying the Operating Voltage Used in the Design

In CPF, operating voltages are associated with *nominal conditions*.

➤ To specify the operating voltages used in the design, use the create_nominal_condition command:

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

Note: The -pmos_bias_voltage and -nmos_bias_voltage options of create_nominal_condition are irrelevant for a design not using substrate biasing.

The example in Figure 2-4 on page 37 uses only one operating voltage, but you can also specify a nominal condition whose voltage is 0.

```
create_nominal_condition -name off -voltage 0

create_nominal_condition -name on -voltage 1.1
```

Specifying the Static Behavior in each Power Mode

➤ To define the static behavior of the design in a power mode, you need to specify the nominal condition of each power domain in that mode, using the `create_power_mode` command:

```
create_power_mode -name string

    -domain_conditions domain_condition_list
    [-default]
```

Note: The -`group_modes` option of the `create_power_mode` command is only relevant for a hierarchical flow. For more information, refer to Chapter 4, “Hierarchical Flow.”

Use the following format to specify a *domain condition* (association of a power domain with its nominal condition in the power mode being defined):

domain_name@nominal_condition_name

For the example in Figure 2-4 on page 37:

```
create_power_mode -name PM1 -domain_conditions {PD1@on PD2@on PD3on} -default
create_power_mode -name PM2 -domain_conditions {PD1@on PD3@on}
create_power_mode -name PM3 -domain_conditions {PD1@on}
```

Note: CPF requires that one power mode is specified as the default power mode.

Note: When a domain is not specified in the list of domain conditions, it is considered to be switched off in the specified mode. For example, power domain `PD2` is not specified in the list of conditions for power mode `PM2`, but referring to Table 2-2 on page 38, power domain `PD2` is switched off in this mode. It is recommended not to rely on the default behavior and to specify all power domains when defining a power mode.

Specifying the Libraries to Use for a Condition

You already grouped libraries that are characterized for a specific set of operating conditions in a library set.

➤ To specify which library set to use for a specific nominal condition, use the `update_nominal_condition` command:

```
update_nominal_condition -name condition
    -library_set library_set
```

For the example in Figure 2-4 on page 37:

```
update_nominal_condition -name on -library_set set1_wc
```

Specifying the Rules to Create Isolation Logic

➤ To define when isolation cells must be added or to specify which pins must be isolated, use the create_isolation_rule command:

```
create_isolation_rule
    -name string
    [-isolation_condition expression | -no_condition]
    {-pins pin_list | -from power_domain_list | -to power_domain_list}...
    [-exclude pin_list]
    [-isolation_target {from|to}]
    [-isolation_output { high | low | hold | tristate}]
    [-secondary_domain power_domain]
```

Typically, isolation logic is needed to isolate signals going from a power domain being switched down to a power domain that remains on. If an input of a powered down domain requires a stable signal for electrical reasons, isolation is required even if the signal goes from a powered on domain to a powered down domain.

Referring to Table 2-2 on page 38, isolation logic will be needed in power modes 2 and 3 for any nets going from power domain PD2 to PD1 and PD3, and for any nets going from power domains PD3 and PD2 to PD1.

For the example in Figure 2-4 on page 37:

```
create_isolation_rule -name iso1 -from PD2 \
-isolation_condition {pm_inst.ice_enable[0]}
create_isolation_rule -name iso2 -to PD1\
-isolation_condition {pm_inst.ice_enable[1]} -isolation_output high
```

In this example, the secondary power domain was not specified for the isolation instances. In this case, the tools will use the power domain of the logic that drives the enable pins. That logic is the pm_inst instance which belongs to power domain PD1.

Specifying the Rules to Create State Retention Logic

➤ To define the rule for replacing selected registers or all registers in the specified power domain with state retention registers, use the create_state_retention_rule command.

```
create_state_retention_rule
    -name string
    { -domain power_domain | -instances instance_list }
    [ -exclude instance_list ]
    [ -restore_edge expr | -save_edge expr
    | -restore_edge expr -save_edge expr
    | -restore_level expr -save_level expr ]
    [ -restore_precondition expr] [-save_precondition expr]
    [-target_type {flop|latch|both}]
    [-secondary_domain domain]
```

For the example in Figure 2-4 on page 37:

```
create_state_retention_rule -name st1 -domain PD2 \
-restore_edge {!pm_inst.pge_enable[0]}
create_state_retention_rule -name st2 -domain PD3 \
-restore_edge {!pm_inst.pge_enable[1]}
```

In this example, the secondary power domain was not specified for the state retention logic. In this case, the tools will use the secondary (or base) power domain of its primary domain. That logic is the `pm_inst` instance which belongs to power domain `PD1`.

Specifying Power Constraints

Note: This information is optional in the CPF file.

➤ To specify the targets for leakage and dynamic power in the current design, use the set_power_target command:

```
set_power_target
    { -leakage float | -dynamic float
    | -leakage float -dynamic float }
```

For the example in Figure 2-4 on page 37:

```
set_power_target -leakage 30 -dynamic 250
```

Specifying Timing Constraints

Note: This information is optional in the CPF file.

➤ To specify the timing constraints for the current design, use the `-sdc_files` option of the `update_power_mode` command:

```
update_power_mode -name mode
    {-sdc_files | -setup_sdc_files | -hold_sdc_files} sdc_file_list
```

For the example in Figure 2-4 on page 37:

```
update_power_mode -name PM1 -sdc_files pm1.sdc
update_power_mode -name PM2 -sdc_files pm2.sdc
```

Specifying Activity Information

Note: This information is optional in the CPF file.

➤ To specify the activity information that can be used for power analysis, use

```
update_power_mode -name mode
    -activity_file file -activity_file_weight weight
```

Supported formats for the activity files are VCD, TCF, and SAIF.

If the design has several modes, you can specify the relative weight of the activities per mode.

For the example in Figure 2-4 on page 37, only the activity file for the default power mode is specified:

```
update_power_mode -name PM1 -activity_file top.tcf -activity_file_weight 100
```

Updating the Rules with Information for Implementation

1. To specify the location or the type of isolation cells to be used, use the update_isolation_rules command:

```
update_isolation_rules -names rule_list
    { -location {from | to | parent | any}
    | -within_hierarchy instance
    | -cells cell_list
    | -prefix string
    | -open_source_pins_only}...
```

2. To append the specified rules for state retention logic with implementation information, use the update_state_retention_rules command:

```
update_state_retention_rules
    -names rule_list
    { -cell_type string
    | -cells cell_list| -set_reset_control} ...
```

   Note: This information is optional in the CPF file.

For the example in Figure 2-4 on page 37:

```
update_isolation_rules -names iso1 -location to -cells ISOLNX2M
update_isolation_rules -names iso2 -location to -cells ISOLNX2M
```

Specifying the Power and Ground Nets

➤ To declare (or create) the nets connected to the power supplies, use the create_ground_nets and create_power_nets commands:

```
create_ground_nets -nets net_list
    [-voltage string]
    [-external_shutoff_condition expression | -internal]
    [-user_attributes string_list]
    [-peak_ir_drop_limit float]
    [-average_ir_drop_limit float]

create_power_nets -nets net_list
    [-voltage string]
    [-external_shutoff_condition expression | -internal]
    [-user_attributes string_list]
    [-peak_ir_drop_limit float]
    [-average_ir_drop_limit float]
```

Note: Power and ground nets referenced in an `update_power_domain` command can be either always on or can be switchable power nets depending on the power domain specification. Other power and ground nets are considered to be always on unless you specify the `-external_shutoff_condition` option.

For the example in Figure 2-4 on page 37,

```
create_power_nets -nets VDD -voltage 1.1
create_power_nets -nets {VDD_SW1 VDD_SW2} -internal
create_ground_nets -nets VSS -voltage 0
```

Specifying the Global Connections

➤ To specify how to connect global nets, such as power and ground nets, use the create_global_connection command:

```
create_global_connection
    -net net
    -pins pin_list
    [-domain domain | -instances instance_list]
```

For the example in Figure 2-4 on page 37:

```
create_global_connection -net VDD -pins VDD
create_global_connection -net VSS -pins VSS
```

Specifying the Rules for the Power Switch Logic

1. To specify how a single power switch must connect the external and internal power or ground nets for the specified power domain, use the create_power_switch_rule command.

```
create_power_switch_rule
    -name string
```

```
    -domain power_domain
    {-external_power_net net | -external_ground_net net}
```

You can specify one or more commands for a power domain depending on whether you want to control the switchable power domain by a single switch or multiple switches.

2. To append the specified rules for power switch logic with implementation information, use the update_power_switch_rule command:

```
update_power_switch_rule
    -name string
    { -enable_condition_1 expression [-enable_condition_2 expression]
    | -acknowledge_receiver_1 express [-acknowledge_receiver_2 express]
    | -cells cell_list
    | -gate_bias_net power_net
    | -prefix string
    | -peak_ir_drop_limit float
    | -average_ir_drop_limit float } ..
```

For the example in Figure 2-4 on page 37:

```
create_power_switch_rule -name SW1 -domain PD2 -external_power_net VDD
create_power_switch_rule -name SW2 -domain PD3 -external_power_net VDD
update_power_switch_rule -name SW1 -cells hd32M -prefix CDN_
update_power_switch_rule -name SW2 -cells hd32M -prefix CDN_
```

Specifying Additional Information for Power and Ground Routing

➤ To specify additional information that applies to power and ground routing, use

```
update_power_domain
    -name domain
    { -primary_power_net net | -primary_ground_net net} ...
```

For the example in Figure 2-4 on page 37:

```
update_power_domain -name PD1 -primary_power_net VDD -primary_ground_net VSS
update_power_domain -name PD2 -primary_power_net VDD_SW1 -primary_ground_net VSS
update_power_domain -name PD3 -primary_power_net VDD_SW2 -primary_ground_net VSS
```

Specifying the Operating Corners

The design must be able to perform under different sets of operating conditions (process, voltage, and temperature values). Because the timing and power characteristics of the cells depend on the operating conditions, different library sets that contain the characterization information for the different conditions are used. An operating corner shows which library set to use for a given operating condition.

Tip

Different portions of the design can operate at the same voltage and yet use dedicated library sets. In this case, it is possible to have multiple operating corners with the same operating conditions. Table 2-4 on page 41 illustrates this case.

➤ To define an operating corner, use the `create_operating_corner` command.

```
create_operating_corner
    -name string
    -voltage float [-ground_voltage float]
    [-process float]
    [-temperature float]
    -library_set library_set
```

Note: The `-pmos_bias_voltage` and `-nmos_bias_voltage` options of `create_operating_corner` are irrelevant for a design not using substrate biasing.

For the design in Figure 2-4 on page 37, two operating corners were specified.

The following command specifies to use library set `set1_wc` for operating corner `WC`. The process value of the operating corner is 1, the temperature is 125°C and the operating voltage is 0.99V. This operating corner is defined for worst case conditions.

```
create_operating_corner -name WC \
    -process 1 -temperature 125 -voltage 0.99 -library_set set1_wc
```

Specifying the Analysis Views

The design must function correctly in each power mode not only under typical conditions, but also under extreme conditions. Typically a multi-mode multi-corner timing analysis will be done for the worst case and the best case conditions.

An analysis view associates a specific operating corner with each power domain in the specified power mode. You need to make sure that all operating corners for a view correspond to either best or worst case conditions.

➤ To define an analysis view, use the `create_analysis_view` command.

```
create_analysis_view
    -name string
    -mode mode
    -domain_corners domain_corner_list
    [-user_attributes string_list]
```

Note: The `-group_views` option of `create_analysis_view` is only relevant in a hierarchical flow. For more information, refer to Chapter 4, “Hierarchical Flow.”

Use the following format to specify a domain corner:

domain_name@corner_name

Tip

The CPF file can contain several analysis views with the same domain and corner information for a power mode. For a multi-mode multi-corner analysis, some implementation and timing analysis tools need unique views to associate with different parasitic corners.

For the design in Figure 2-7 on page 64, six analysis views were specified.

The following command specifies analysis view `AV_PM1` for power mode `PM1`. For this view the operating corners for the best case operating conditions are associated with the three power domains.

```
create_analysis_view -name AV_PM1 -mode PM1 \
    -domain_corners {PD1@BC PD2@BC PD3@BC}
```

Creating a CPF File for a Design Using DVFS Methodology

Dynamic voltage frequency scaling (DVFS) reduces the power in the chip by scaling down the voltage and frequency when peak performance is not required.

A design using DVFS can be seen as a special case of an MSV design operating in multiple design modes.

- In a pure MSV design different portions of the design operate on different voltages and these portions *remain* operating at their respective operating voltage.
- In a DVFS design, in addition some portions can dynamically *change* to other voltages depending on the design mode or can even be switched off.

  Consequently, a DVFS design must satisfy different constraints in different design modes.

Requirements for DVFS Designs

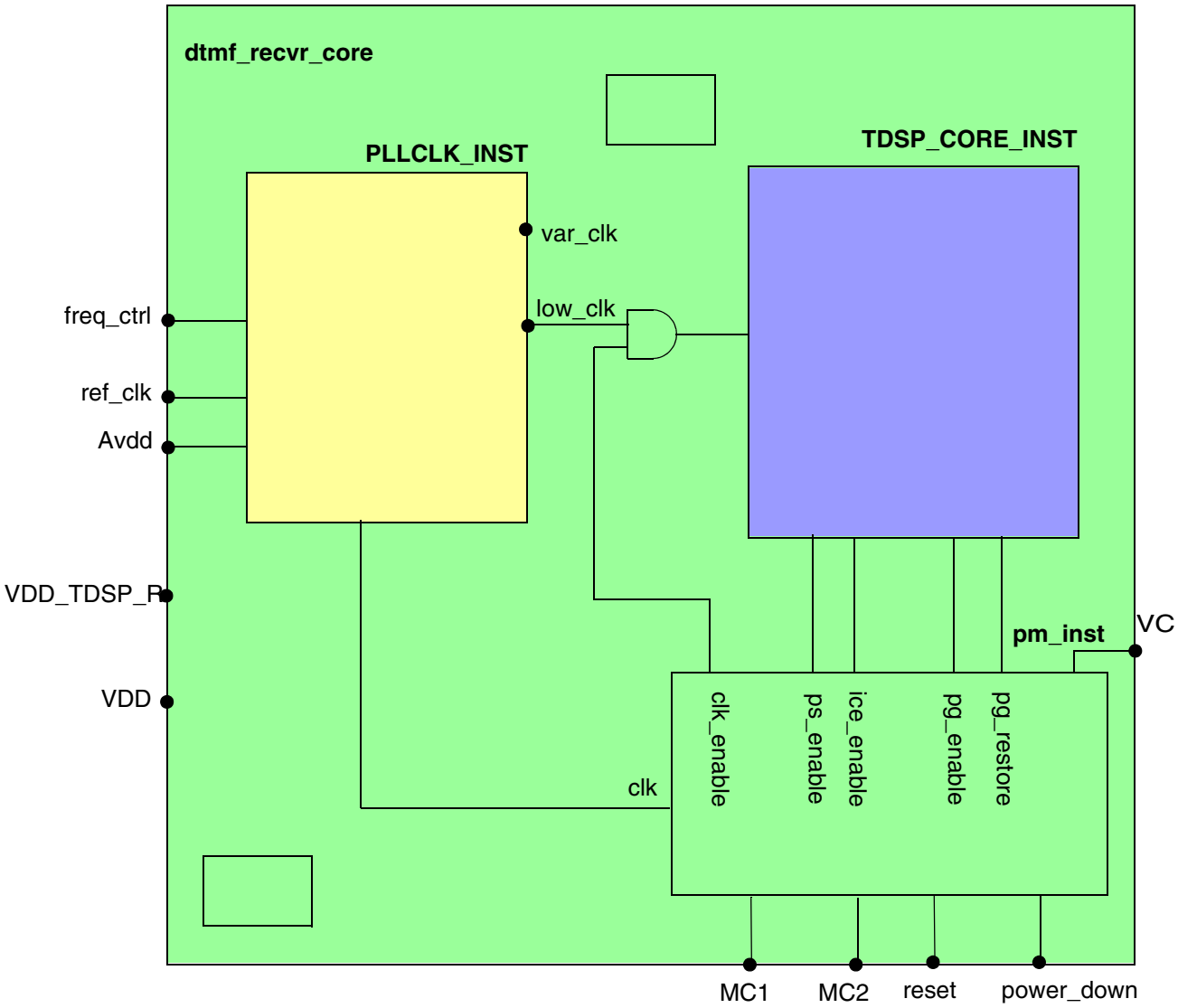
DVFS designs require variable power supply(ies) that can generate the required voltage levels with minimal transition energy losses and a quick voltage transient response.

When scaling the voltage, the frequency must be scaled accordingly to meet signal propagation delay requirements.

A power scheduler can intelligently compute the appropriate frequency and voltage levels needed to execute the various applications.

Figure 2-7 Example of DVFS Design

Instances Operating on same Operating Voltage	Corresponding Power Domain	Libraries used for all modes
dtmf_recvr_core, pm_inst	AO	ao_bc_0v99, ao_wc_0v99 ao_bc_0v792, ao_wc_0v792
TDSP_CORE_INST	TDSPCORE	tdsp_bc_0v792, tdsp_wc_0v792
PLLCLK_INST	PLL	ao_bc_0v99, ao_wc_0v99



The dtmf_recvr_core design shown in Figure 2-7 on page 64 has several other blocks which for the sake of simplicity are not shown here. However, they all operate at the same voltage as the top-level of the design.

The voltage of the top-level design is scaled depending on the requested function of the design. If the processing speed is critical a higher voltage is used, if the processing speed is not critical, the voltage is dynamically scaled down together with the clock frequency to save power. For this design we assume that the voltage supply is dynamically controlled external to the chip. The input power signal for the top-level and the blocks that operate at the same voltage is VDD.

Note: The design used here uses both DVFS and PSO methodology.

The dtmf_recvr_core design further contains

- The TDSP_CORE_INST block

  This digital signal processing block operates at a lower voltage because its processing speed is not critical. When the block does not need to be operational, it is shut down.

  The power input for this block is Vdd_TDSP_R. The clk_enable signal disables the clock when the block is shut down.

- The PLLCLK_INST block

  This block is used to generate the clocks needed by all the blocks in the design. It has a reference clock, ref_clk, that is used to generate all other clocks.

  Because the design uses two operating voltages, two clock signals are created:

  - The low_clk clock signal which feeds the TDSP_CORE_INST block has a constant lower frequency.
  - The var_clk clock signal feeds the top-level design and other blocks and can vary in clock frequency depending on the operating voltage.

  The freq_ctrl signal ensures that the frequency of the var_clk signal used for the top-level design is scaled proportional to the voltage.

  Because this block is an analog block, it needs to operate at a constant voltage to ensure correct working. It therefore needs a dedicated power input, Avdd.

- The pm_inst block

  This block generates all power control signals for the chip.

  This block operates at the same voltage as the top-level of the design.

In DVFS designs, a collection of logic blocks (hierarchical instances) and leaf instances that use the same *main* power supply and whose voltage and frequency can *simultaneously* change or be switched off belong to the same *power domain.*

The example design in Figure 2-7 on page 64 has the following power domains.

- The `PLLCLK_INST` block is the only block in the design that operates at constant voltage 0.99V. This block belongs to power domain `PLL`.
- The `TDSP_CORE_INST` block operates at voltage 0.792V and it is the only block that is shut down at certain times. This block belongs to power domain `TDSPCORE`.
- The `pm_inst` block, the top-level design and the remaining blocks are always powered on but their operating voltage and frequency can change. They belong to power domain `AO`.

Power domains `PLL` and `AO` are never powered down. They are referred to as an unswitched domain. Power domain `TDSPCORE` can be powered down and is called a switchable domain. CPF distinguishes between internal switchable, and on-chip controlled external switchable domains.

A steady state of a design in which some power domains are switched on and some power domains are switched off is called a ***power mode***. In a power mode, each power domain operates on a specific voltage (nominal condition). Table 2-5 shows the operating voltages for each of the power domains in the three power modes of the `dtmf_recvr_core` design. The voltages shown in this table correspond to the worst case voltages. The typical voltages for the design would be 1.1V and 0.88V.

Table 2-5 Power Modes

Power Mode	Corresponding Power Domain		
	AO	PLL	TDSPCore
full	0.99	0.99	0.792
slow	0.99	0.99	0.0
sleep	0.792	0.99	0.0

To pass signals between portions of the design that operate on different voltages, ***level shifters*** are needed.

To prevent unknown states from propagating from a power domain that is powered-down to a power domain that remains on, ***isolation cells*** are needed at the boundaries of the power domains that are powered down. Most of the time, isolation cells are inserted at the output

boundaries of the powered down domains. You can, however, also insert isolation cells at the input boundaries.

To facilitate powered down blocks to resume normal operation, ***state retention cells*** can be used for some sequential cells to keep their previous state prior to power up.

To connect and disconnect the power supply from the gates in a power domain, you must add ***power switch logic*** or use an external power shut-off method.

For switchable domains you need to indicate how the power supply is connected and disconnected from the gates.

- For internal switchable domains, you must add ***power switch logic.***
- For external switchable domains, the power switch logic is not part of the chip, so you must indicate that an *external power shut-off method* is used.

For this example we are assuming that power domain `TDSPCore` is internal switchable.

Special control signals are used to control the supply voltage, shut down a power domain, enable state retention, restore the state of the registers when powering up a power domain, and control the working of the power switch logic. Table 2-6 shows the signals used in this design example.

Table 2-6 Signals Controlling the Power Domains

Power Domain	Control Signals			
	voltage control	power switch	isolation cell	state retention cell
AO	VC	no control signal	no control signal	no control signal
PLL	no control signal	no control signal	no control signal	no control signal
TDSPCore	no control signal	ps_enable	iso_enable	pg_enable and pg_restore

When a domain is switchable, it derives its power from another power domain through either internal or external power switch logic.

In this example, power domain `TDSPCore` derives its power from power domain `AO`, then `AO` is called the ***secondary*** (or base) domain for `TDSPCore`, which is referred to as the ***primary*** (or derived) domain.

When defining a (primary) power domain you can indicate its secondary domain and under which condition the domain will be shut down.

The majority of instances in a power domain are driven by the same power supply. For switchable domains, it is the primary power and ground nets of the (primary) power domain to which the instances belong that provide the power supply to the power and ground pins (follow-pins) of the cell.

On the other hand, level shifters, isolation cells and state retention cells are driven by multiple power supplies. These special low power instances have at least two sets of power and ground pins, and are therefore associated with two power domains.

For level shifters, the primary and a secondary power domain are defined as follows:

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

The tools reading CPF can derive the primary and secondary power domains for level shifters. For more information, refer to Input and Output Domains of Level Shifters.

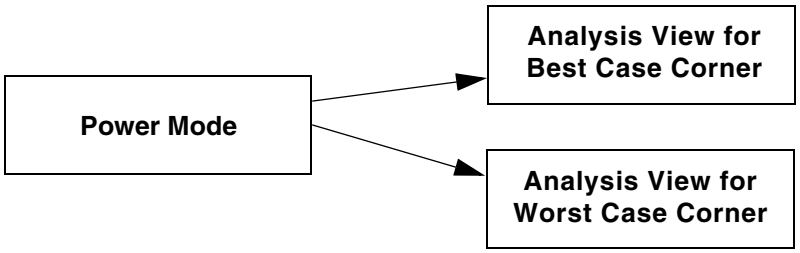
For isolation cells and state retention cells the primary and a secondary power domain are defined as follows:

- the ***primary*** domain is the domain that provides the power supply to their primary set of power and ground pins.
- The ***secondary*** domain is the domain whose primary power and ground nets provide the power supply to the secondary power and ground pins of the isolation cells and state retention cells.

For isolation cells and state retention cells, it is recommended that you specify the secondary domain for the isolation cells and state retention cells, but if you do not specify this information, the tools can use some rules to derive the secondary domain. For more information, refer to Secondary Power Domain of Isolation Instances, and Secondary Domain of Retention Logic.

When the design can operate in different power modes, you need to check if the design functions correctly in each of these modes not only at the typical conditions but also when slightly different operating conditions apply. Typically a multi-mode multi-corner timing analysis will be done for the worst case and the best case corner. The design must function correctly in both corners for the same set of timing constraints that is specified for that power mode. Figure 2-8 on page 69 shows that an analysis (view) can be done for each corner of the power mode.

Figure 2-8 Multi-Corner Timing Analysis for a Power Mode



When analyzing a view of the design, you need to indicate the corners at which the power domains are operating, as shown in Figure 2-9.

Figure 2-9 Relation of Analysis View with Power Domains and Operating Corners

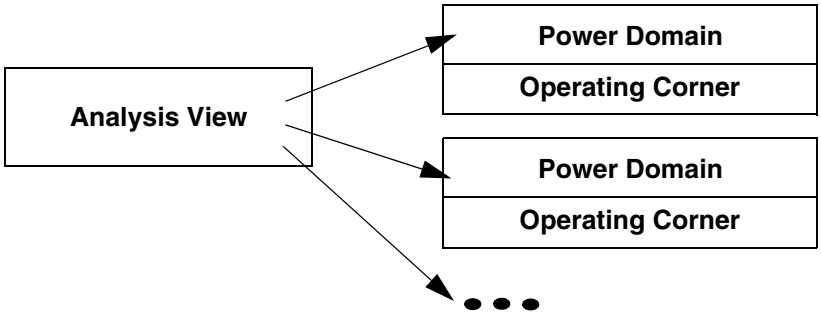


Table 2-7 indicates which library sets must be used for this design to check the conditions.

Table 2-7 Operating Corners

Operating Corner	Operating Condition			Library set
Name	process	temperature	voltage	
BC08COM_AO	1	0	0.968	ao_bc_0v792
BC08COM_TDSP	1	0	0.968	tdsp_bc_0v792
BCCOM_AO	1	0	1.21	ao_bc_0v99
WC08COM_AO	1	125	0.792	ao_wc_0v792
WC08COM_TDSP	1	125	0.792	tdsp_wc_0v792
WCCOM_AO	1	125	0.99	ao_wc_0v99

As is obvious that all this information is related to the power intent of the design, it makes

sense to describe this information in the CPF file.

Complete CPF File for DVFS Example

```
set_cpf_version 1.1
################################################
#       Technology part of the CPF
################################################
set libdir ../LIBS
set lib_0v99_wc " $libdir/timing/tcbn45lpbwp_c060907wc.lib "
set lib_ao_wc " $libdir/timing/tcbn45lpbwp_wc0d720d9.lib \
                $libdir/timing/pllclk_slow.lib \
                $libdir/timing/ram_256x16A_slow.lib \
                $libdir/timing/rom_512x16A_slow.lib "
set lib_0v99_bc " $libdir/N45/timing/tcbn45lpbwp_c060907bc.lib "
set lib_ao_bc " $libdir/timing/tcbn45lpbwp_bc0d881d1.lib \
                $libdir/timing/pllclk_slow.lib \
                $libdir/timing/ram_256x16A_slow.lib \
                $libdir/timing/rom_512x16A_slow.lib "
set lib_0v792_wc " $libdir/timing/tcbn45lpbwp_c060907wc0d72.lib "
set lib_tdsp_wc " $libdir/timing/tcbn45lpbwp_wc0d90d72.lib \
                  $libdir/timing/tcbn45lpbwphvt_wc0d72.lib \
                  $libdir/timing/tcbn45lpbwp_wc0d72_ptlvl.lib "
set lib_0v792_bc " $libdir/timing/tcbn45lpbwp_c060907bc0d88.lib "
set lib_tdsp_bc " $libdir/timing/tcbn45lpbwp_bc1d10d88.lib
                  $libdir/timing/tcbn45lpbwphvt_bc0d88.lib \
                  $libdir/timing/tcbn45lpbwp_bc0d88_ptlvl.lib "
# define the library sets
define_library_set -name ao_wc_0v99 -libraries "$lib_0v99_wc $lib_ao_wc"
define_library_set -name ao_bc_0v99 -libraries "$lib_0v99_bc $lib_ao_bc"
define_library_set -name ao_wc_0v792 -libraries "$lib_0v792_wc $lib_ao_wc"
define_library_set -name ao_bc_0v792 -libraries "$lib_0v792_bc $lib_ao_bc"
define_library_set -name tdsp_wc_0v792 -libraries "$lib_0v792_wc $lib_tdsp_wc"
define_library_set -name tdsp_bc_0v792 -libraries "$lib_0v792_bc $lib_tdsp_bc"
# define the level shifters
define_level_shifter_cell -cells LVL*HLD* \
    -input_voltage_range 0.792:0.99:0.099 \
    -output_voltage_range 0.792:0.99:0.099 \
    -direction down \
    -output_power_pin VDD \
    -ground VSS \
    -valid_location to
define_level_shifter_cell -cells PTLVL*HLD* \
    -input_voltage_range 0.792:0.99:0.099 \
    -output_voltage_range 0.792:0.99:0.099 \
    -direction down \
    -output_power_pin TVDD \
    -ground VSS \
    -valid_location to
```

```
define_level_shifter_cell -cells LVLLHD* \
    -input_voltage_range 0.792:0.99:0.099 \
    -output_voltage_range 0.792:0.99:0.099 \
    -input_power_pin VDDL \
    -output_power_pin VDD \
    -direction up \
    -ground VSS \
    -valid_location to
```

define the enable level-shifter cell

```
define_level_shifter_cell -cells LVLLHCD* \
    -input_voltage_range 0.792:0.99:0.099 \
    -output_voltage_range 0.792:0.99:0.099 \
    -enable NSLEEP \
    -input_power_pin VDDL \
    -output_power_pin VDD \
    -direction up \
    -ground VSS \
    -valid_location to
```

define the isolation cells

```
define_isolation_cell -cells iso* \
    -power VDD \
    -ground VSS \
    -enable NSLEEP \
    -valid_location to
```

define the always on cell

```
define_always_on_cell -cells {PTBUFFD2BWP} \
    -power_switchable VDD -power TVDD -ground VSS
```

define the state retention cell

```
define_state_retention_cell -cells { RSDFCSRHD2BWP } \
    -clock_pin CP \
    -power TVDD \
    -power_switchable VDD \
    -ground VSS \
    -save_function "SAVE" \
    -restore_function "!NRESTORE"
```

define the power switch cells

```
define_power_switch_cell -cells {HDRDID1BWPHVT HDRDIAOND1BWPHVT} \
    -power_switchable VDD -power TVDD \
    -stage_1_enable !NSLEEPIN1 \
    -stage_1_output NSLEEPOUT1 \
    -stage_2_enable !NSLEEPIN2 \
    -stage_2_output NSLEEPOUT2 \
    -type header


################################################
#       Design part of the CPF
################################################
set_design dtmf_recvr_core
set_time_unit ms
set_hierarchy_separator "/"
set constraintDir ../mmmc
```

```
# create nominal conditions
create_nominal_condition -name high_ao -voltage 0.99
create_nominal_condition -name low_ao -voltage 0.792
create_nominal_condition -name low_tdsp -voltage 0.792
create_nominal_condition -name off -voltage 0
# create power domains
create_power_domain -name AO -default \
    -active_state_conditions {low_ao@"!VC" high_ao@"VC"}
create_power_domain -name TDSPCore -instances TDSP_CORE_INST \
    -shutoff_condition {PM_INST/ps_enable} -base_domains AO
create_power_domain -name PLL -instances PLLCLK_INST \
    -boundary_ports {refclk vcom vcop ibias pllrst}
# create power modes
create_power_mode -name full \
    -domain_conditions {AO@high_ao PLL@high_ao TDSPCore@low_tdsp} -default
create_power_mode -name slow \
    -domain_conditions {AO@high_ao PLL@high_ao TDSPCore@off}
create_power_mode -name sleep \
    -domain_conditions {AO@low_ao PLL@high_ao TDSPCore@off}
# create rules for level shifter insertion
create_level_shifter_rule -name LSRULE_H2L -from AO -to TDSPCore \
    -exclude {PM_INST/ps_enable PM_INST/pg_enable PM_INST/pg_restore}
create_level_shifter_rule -name LSRULE_H2L_AO -from AO -to TDSPCore \
    -pins {PM_INST/ps_enable PM_INST/pg_enable PM_INST/pg_restore}
create_level_shifter_rule -name LSRULE_H2L_PLL -from PLL -to AO
# create rule for isolation logic insertion
create_isolation_rule -name ISORULE -from TDSPCore  \
    -isolation_condition "!PM_INST/iso_enable" -isolation_output high
# create rule for state retention insertion
create_state_retention_rule -name SRPG_TDSP \
    -domain TDSPCore \
    -restore_edge {!PM_INST/pg_restore} \
    -save_edge {PM_INST/pg_enable}
####################################################
# Optional Information for RTL Simulation
####################################################
update_power_domain -name AO -transition_latency {low_ao high_ao@0.8:1.2}
update_power_domain -name AO -transition_latency {high_ao low_ao@0.2:0.3}
update_power_domain -name TDSPCore -transition_latency {off low_tdsp@2.0:2.5}
####################################################
# Additional Information for Logic Synthesis
####################################################
# associate library sets with nominal conditions
update_nominal_condition -name high_ao -library_set ao_wc_0v99
update_nominal_condition -name low_ao -library_set ao_wc_0v792
update_nominal_condition -name low_tdsp -library_set tdsp_wc_0v792
```

```
# specify timing constraints
update_power_mode -name full \
    -sdc_files ${constraintDir}/dtmf_recvr_core_gate.sdc
update_power_mode -name slow \
    -sdc_files ${constraintDir}/dtmf_recvr_core_gate.sdc
update_power_mode -name sleep \
    -sdc_files ${constraintDir}/dtmf_recvr_core_dull.sdc
# update the rules
update_level_shifter_rules -names LSRULE_H2L -cells LVLHLD2BWP -location to
update_level_shifter_rules -names LSRULE_H2L_AO -cells PTLVLHLD2BWP -location to
update_level_shifter_rules -names LSRULE_H2L_PLL -cells LVLHLD2BWP -location to
update_isolation_rules -names ISORULE -location to -cells LVLLHCD2BWP
update_state_retention_rules -names SRPG_TDSP \
    -cell RSDFCSRHD2BWP -library_set tdsp_wc_0v792
#####################################################
# Additional Information for Physical Implementation
#####################################################
# declare power and ground nets
create_power_nets -nets VDD -voltage {0.792:0.99:0.198}
create_power_nets -nets VDD_TDSP_R -voltage 0.792
create_power_nets -nets Avdd -voltage 0.99
create_power_nets -nets VDD_TDSPCore -internal -voltage 0.792
create_ground_nets -nets VSS
create_ground_nets -nets Avss
# create global connections
create_global_connection -domain AO -net VDD_TDSPCore -pins VDDL
create_global_connection -domain AO -net VDD -pins VDD
create_global_connection -domain AO -net VSS -pins VSS
create_global_connection -domain PLL -net Avdd -pins avdd!
create_global_connection -domain PLL -net Avss -pins agnd!
create_global_connection -domain PLL -net VDD -pins VDDL
create_global_connection -domain PLL -net Avdd -pins VDD
create_global_connection -domain PLL -net Avss -pins VSS
create_global_connection -domain TDSPCore -net VSS -pins VSS
create_global_connection -domain TDSPCore -net VDD_TDSP_R -pins TVDD
create_global_connection -domain TDSPCore -net VDD_TDSPCore -pins VDD
# rule for power switch insertion
create_power_switch_rule -name TDSPCore_SW -domain TDSPCore \
    -external_power_net VDD
update_power_switch_rule -name TDSPCore_SW -cells HDRDID1BWPHVT \
    -prefix CDN_SW_ -acknowledge_receiver_1 switch_en_out
# add implementation info for power domains
update_power_domain -name AO -primary_power_net VDD -primary_ground_net VSS
```

```
update_power_domain -name TDSPCore -primary_power_net VDD_TDSPCore \
-primary_ground_net VSS

update_power_domain -name PLL -primary_power_net Avdd -primary_ground_net Avss

# create operating corners

create_operating_corner -name BCCOM_AO \
    -process 1 -temperature 0 -voltage 1.21 -library_set ao_bc_0v99

create_operating_corner -name WCCOM_AO \
    -process 1 -temperature 125 -voltage 0.99 -library_set ao_wc_0v99

create_operating_corner -name BC08COM_AO \
    -process 1 -temperature 0 -voltage 0.968 -library_set ao_bc_0v792

create_operating_corner -name BC08COM_TDSP \
    -process 1 -temperature 0 -voltage 0.968 -library_set tdsp_bc_0v792

create_operating_corner -name WC08COM_AO \
    -process 1 -temperature 125 -voltage 0.792 -library_set ao_wc_0v792

create_operating_corner -name WC08COM_TDSP \
    -process 1 -temperature 125 -voltage 0.792 -library_set tdsp_wc_0v792

# create analysis views

create_analysis_view -name AV_full_MIN_RC1 -mode full \
    -domain_corners {AO@BCCOM_AO PLL@BCCOM_AO TDSPCore@BC08COM_TDSP}

create_analysis_view -name AV_full_MIN_RC2 -mode full \
    -domain_corners {AO@BCCOM_AO PLL@BCCOM_AO TDSPCore@BC08COM_TDSP}

create_analysis_view -name AV_full_MAX_RC1 -mode full \
    -domain_corners {AO@WCCOM_AO PLL@WCCOM_AO TDSPCore@WC08COM_TDSP}

create_analysis_view -name AV_full_MAX_RC2 -mode full \
    -domain_corners {AO@WCCOM_AO PLL@WCCOM_AO TDSPCore@WC08COM_TDSP}

create_analysis_view -name AV_slow_MIN_RC1 -mode slow \
    -domain_corners {AO@BCCOM_AO PLL@BCCOM_AO TDSPCore@BC08COM_TDSP}

create_analysis_view -name AV_slow_MAX_RC1 -mode slow \
    -domain_corners {AO@WCCOM_AO PLL@WCCOM_AO TDSPCore@WC08COM_TDSP}

create_analysis_view -name AV_sleep_MIN_RC1 -mode sleep \
    -domain_corners {AO@BC08COM_AO PLL@BCCOM_AO TDSPCore@BC08COM_TDSP}

create_analysis_view -name AV_sleep_MAX_RC1 -mode sleep \
    -domain_corners {AO@WC08COM_AO PLL@WCCOM_AO TDSPCore@WC08COM_TDSP}

end_design
```

Steps to Create the CPF File for DVFS Design

This section describes the information to include in a CPF file for a design using the DVFS methodology. The example shown in Figure 2-7 on page 64 is used throughout this section.

For a design using the DVFS methodology, the technology-related information lists the libraries that you want to use for the design and identifies the library cells that can be used as level shifters, isolation cells, always-on cells, power switch cells and state retention cells.

- Specifying the Libraries on page 78
- Specifying the Level Shifter Cells to Use on page 79
- Specifying the Isolation Cells to Use on page 80
- Specifying the Always-On Cells on page 81
- Specifying the State Retention Cells to be Used on page 82
- Specifying the Power Switch Cells to be Used on page 83

The design-related information captures the power intent and constraints.

The following information is needed for both design creation and logic verification:

- Declaring the Design Described in the CPF File on page 84
- Specifying the Units Used on page 84
- Specifying the Naming Styles Used on page 85
- Specifying the Operating Voltages Used in the Design on page 86
- Specifying the Power Domains on page 87
- Specifying the Static Behavior in each Power Mode on page 88
- Specifying the Rules to Create Level-Shifter Logic on page 89
- Specifying the Rules to Create Isolation Logic on page 90
- Specifying the Rules to Create State Retention Logic on page 91
- Specifying the Time to Transition between Power States on page 92
- Specifying the Power Constraints on page 93
- Specifying the Timing Constraints on page 93
- Specifying the Activity Information on page 93

- Updating the Rules with Implementation Information on page 94

The following information is needed for physical implementation:

- Specifying the Global Power and Ground Nets on page 95
- Specifying the Global Connections on page 96
- Specifying the Rules for the Power Switch Logic on page 97
- Specifying Additional Information for Power and Ground Routing on page 98
- Specifying the Operating Corners on page 99
- Specifying the Analysis Views on page 100

Specifying the Libraries

➤ To group libraries that are characterized for a specific set of operating conditions, use the define_library_set command:

```
define_library_set -name library_set
    -libraries library_list
```

For the design in Figure 2-7 on page 64, six library sets are defined: one library set for the best and worst case operating conditions of each power domain.

The following command defines the library set to be used for the main portion of the design (AO power domain). The libraries were characterized for the worst condition. The command uses two variables defined in the CPF file to pass two library lists.

```
define_library_set -name ao_wc_0v99 -libraries "$lib_0v99_wc $lib_ao_wc"
```

Specifying the Level Shifter Cells to Use

➤ To identify which cells in the libraries must be used as level shifters, use the `define_level_shifter_cell` command.

For more information on how to model different types of level shifters, refer to Modeling Level Shifters on page 150.

For applications that do not read .lib files, you must specify the library cells to allow the application to identify the instances of these cells in the netlist.

When you define the input_voltage_range for a pure MSV design, the value for the `-input_voltage_range` and `-output_voltage_range` options is a single voltage value. For DVFS designs, the level shifters must be able to support a range and thus in this case you must specify a range as value for these options:

`lower_bound:upper_bound:step`

For the design in Figure 2-7 on page 64, four groups of level shifters are specified.

The following command selects a group of level shifters whose input and output voltages can range between 0.792V and 0.99V in increments of 0.099V. The cells can only be used from a higher to a lower voltage and must be placed in the destination power domain.

```
define_level_shifter_cell -cells PTLVL*HLD* \
    -input_voltage_range 0.792:0.99:0.099 \
    -output_voltage_range 0.792:0.99:0.099 \
    -direction down
    -output_power_pin TVDD -ground VSS \
    -valid_location to
```

Specifying the Isolation Cells to Use

➤ To identify which cells in the libraries must be used as isolation cells, use the `define_isolation_cell` command.

For more information on how to model different types of isolation cells, refer to Modeling Isolation Cells on page 165.

For applications that do not read .lib files, you must specify these library cells to allow these applications to identify instances of isolation cells in the netlist.

For the design in Figure 2-7 on page 64 isolation cells are needed at the boundaries of the `TDSP_CORE_INST` block. For this design, one command is specified.

The following command selects cells whose enable pin is called `NSLEEP`, and whose valid location is the destination power domain. The command also specifies that the names of the power and ground pins of the corresponding LEF cells is called `VDD` and `VSS`, respectively.

```
define_isolation_cell -cells iso* \
    -power VDD \
    -ground VSS \
    -enable NSLEEP \
    -valid_location to
```

Specifying the Always-On Cells

Always-on cells are special cells whose power supply has to be continuous on even when the power supply for the rest of the logic in the power domain is off.

Always-on cells are used for example

- To drive the control signals of the state retention cells in a domain that is being powered down
- In combination with isolation cells that are inserted in the power domain that is switched off to ensure that the driver of the enable pin of the isolation cells is never switched off.

➤ To identify which cells in the libraries must be used as always-on cells, use the define_always_on_cell command.

```
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 ]
```

Note: Outputs of cells that are always on, are always-on drivers.

For applications that do not read .lib files, you must specify the library cells to allow the application to identify the instances of these cells in the netlist.

For the design in Figure 2-7 on page 64, always-on cells are needed to drive the control signals of the state retention cells in the TDSP_CORE_INST block. For the design, one command is specified.

The following command specifies to use the PTBUFFD2BWP cell. It also specifies that in the corresponding LEF cell, the name of the pin connected to the power that is switched off is VDD, the name of the pin connected to the power that remains on while the power domain is shut off is TVDD, and the name of the pin connected to the ground is VSS.

```
define_always_on_cell -cells {PTBUFFD2BWP} \
    -power_switchable VDD -power TVDD -ground VSS
```

Specifying the State Retention Cells to be Used

➤ To identify which cells in the libraries must be used as state retention cells, use the define_state_retention_cell command.

For more information on how to model different types of state retention cells, refer to Modeling State Retention Cells on page 181.

For applications that do not read .lib files, you must specify the library cells to allow the application to identify the instances of these cells in the netlist.

For the design in Figure 2-7 on page 64, state retention cells are needed for the TDSP_CORE_INST block. For the design one command was specified.

The following command selects the RSDFCSRHD2BWP cell and specifies that clock pin is CP, that the state of the cell is saved when pin SAVE is active high and that the state of the cell will be restored when pin NRESTORE is active low. It also specifies that in the corresponding LEF cell, the name of the pin connected to the power that is switched off is VDD, the name of the pin connected to the power that remains on while the power domain is shut off is TVDD, and the name of the pin connected to the ground is VSS.

```
define_state_retention_cell -cells { RSDFCSRHD2BWP } \
    -clock_pin CP \
    -power TVDD \
    -power_switchable VDD \
    -ground VSS \
    -save_function "SAVE" \
    -restore_function "!NRESTORE"
```

Specifying the Power Switch Cells to be Used

➤ To identify which cells in the libraries must be used as power switch cells, use the define_power_switch_cell command.

For more information on how to model different types of power switch cells, refer to Modeling Power Switch Cells on page 188.

For applications that do not read .lib files, you must specify these library cells to allow these application to identify the instances of these cells in the netlist.

For the design in Figure 2-7 on page 64 power switch cells are needed for the TDSP_CORE_INST block. For the design one command was specified.

The following command specifies to use the HDRDID1BWPHVT and HDRDIAOND1BWPHVT cells. It also specifies that the corresponding LEF cell must be a header cell and that the name of the pin connected to the power that is switched off is VDD and the name of the pin connected to the power that remains on while the power domain is shut off is TVDD. It further specifies that the power switch is turned on when a low value is applied to both the NSLEEPIN1 and NSLEEPIN2 pins and that both output pins are the buffered outputs of the corresponding input pins.

```
define_power_switch_cell -cells {HDRDID1BWPHVT HDRDIAOND1BWPHVT} \
    -power_switchable VDD -power TVDD \
    -stage_1_enable !NSLEEPIN1 \
    -stage_1_output NSLEEPOUT1 \
    -stage_2_enable !NSLEEPIN2 \
    -stage_2_output NSLEEPOUT2 \
    -type header
```

Declaring the Design Described in the CPF File

➤ To identify the design for which the CPF file is created, use the following command:

```
set_design module
```

where *module* refers to the name of the top module of the design to which the power information in the CPF file applies.

For the design in Figure 2-7 on page 64:

```
set_design dtmf_recvr_core
```

➤ To indicate when the power information for this module ends, use the following command:

```
end_design
```

Specifying the Units Used

You can specify the units that will be used for the power and time values in CPF commands.

➤ To specify the power unit used, use the following command

```
set_power_unit [pW|nW|uW|mW|W]
```

Default: mW

➤ To specify the time unit used, use the following command

```
set_time_unit [ns|us|ms]
```

Default: `ns`

Note: These commands are optional if you use the default values.

For the design in Figure 2-7 on page 64 the default value was assumed for the power unit., but the time unit was specified:

```
set_time_unit ms
```

Specifying the Naming Styles Used

➤ To specify the hierarchy separator used in the CPF file, use the following command

`set_hierarchy_separator` [*character*]

The default separator is the period (.).

The format of a name in RTL and in the netlist can be different. When you want to use the RTL names in the CPF file, but you are reading a gate-level netlist, you need to specify how the base name and bit information are represented in the netlist.

➤ To specify the format used to name flip-flops and latches in the netlist starting from the register names in the RTL description, use the following command

`set_register_naming_style` [string%s]

Default: `_reg%s`

➤ To specify the format used to name the design objects in the netlist starting from multi-bit arrays in the RTL description, use the following command

`set_array_naming_style` [*string*]

Default: `\[%d\]`

For more information on these two commands, refer to Individual Registers Names in the *Common Power Format Language Reference.*

Note: These three commands are optional if you use the default values.

For the design in Figure 2-7 on page 64 the hierarchy separator was specified:

```
set_hierarchy_separator "/"
```

Specifying the Operating Voltages Used in the Design

In CPF, operating voltages are associated with *nominal conditions*.

➤ To specify the operating voltages used in the design, use the `create_nominal_condition` command:

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

Note: The `-pmos_bias_voltage` and `-nmos_bias_voltage` options of `create_nominal_condition` are irrelevant for a design not using substrate biasing.

For the design in Figure 2-7 on page 64 four nominal conditions are defined.

The following command defines the nominal condition for the highest voltage used in the design.

```
create_nominal_condition -name high_ao -voltage 0.99
```

Specifying the Power Domains

➤ To identify portions of the design that use the same *main* power supply and whose voltage and frequency can *simultaneously* change or be switched off, associate these portions with a power domain using the create_power_domain command:

```
create_power_domain -name power_domain
    [-instances instance_list] [-boundary_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} ]
    [ -active_state_conditions active_state_condition_list ]
    [ -base_domains domain_list]
```

The -shutoff_condition determines when the power domain is switched off. If this option is not specified, the power domain is an unswitched domain.

Note: CPF requires that the top module belongs to the default power domain.

For the design in Figure 2-7 on page 64 three power domains are created.

The following command defines the AO power domain as the default power domain of the scope, and specifies that the power domain operates at different voltages with different control conditions.

```
create_power_domain -name AO -default \
    -active_state_conditions {low_ao@"!VC" high_ao@"VC"}
```

The following command defines the PLL power domain. It associates the hierarchical instance, PLLCLK_INST, and the I/O ports refclk, vcom, vcop, ibias, and pllrst with this domain. These ports are ports that feed signals that are only needed by the PLLCLK_INST instance.

```
create_power_domain -name PLL -instances PLLCLK_INST \
    -boundary_ports {refclk vcom vcop ibias pllrst}
```

Specifying the Static Behavior in each Power Mode

➤ To define the static behavior of the design in a power mode, you need to specify the nominal condition of each power domain in that mode using the create_power_mode command:

```
create_power_mode -name string
    -domain_conditions domain_condition_list
    [-default]
```

Note: The -group_modes option of the create_power_mode command is only relevant for a hierarchical flow. For more information, refer to Chapter 4, "Hierarchical Flow."

Use the following format to specify a *domain condition* (association of a power domain with its nominal condition in the power mode being defined):

domain_name@*nominal_condition_name*

For the design in Figure 2-7 on page 64 three power modes are defined.

The following command defines power mode, full, which corresponds to the power mode in which the full functionality can be accessed. This command specifies that both power domains AO and PLL are operating at nominal condition high_ao, while power domain TDSPCore is operating at nominal condition low_tdsp.

```
create_power_mode -name full \
    -domain_conditions {AO@high_ao PLL@high_ao TDSPCore@low_tdsp} -default
```

Note: When a domain is not specified in the list of domain conditions, it is considered to be switched off in the specified mode. For example, power domain TDSPCore could be omitted from the list of conditions for power modes slow and sleep, because referring to Table 2-5 on page 66, power domain TDSPCore is switched off in both modes. It is recommended not to rely on the default behavior and to specify all power domains when defining a power mode.

For example the following two commands are equal:

```
create_power_mode -name slow \
    -domain_conditions {AO@high_ao PLL@high_ao TDSPCore@off}
create_power_mode -name slow \
    -domain_conditions {AO@high_ao PLL@high_ao}
```

Specifying the Rules to Create Level-Shifter Logic

Depending on your technology, you may need level shifters when passing any signals

- From a power domain with a lower voltage to a power domain with a higher voltage
- From a power domain with a higher voltage to a power domain with a lower voltage
- In both cases

➤ To create the rule to be used between power domains or a set of pins, use the `create_level_shifter_rule` command:

```
create_level_shifter_rule -name string
    {-pins pin_list | -from power_domain_list | -to power_domain_list}...
    [-exclude pin_list]
```

For the design in Figure 2-7 on page 64, three sets of level shifter rules were created.

The following command creates a rule between power domains `AO` and `TDSPCore` and specifies to exclude the `ps_enable`, `pg_enable`, and `pg_restore` pins on the `pm_instance` block from level-shifter insertion.

```
create_level_shifter_rule -name LSRULE_H2L -from AO -to TDSPCore \
    -exclude {PM_INST/ps_enable PM_INST/pg_enable PM_INST/pg_restore}
```

Specifying the Rules to Create Isolation Logic

➤ To define when isolation cells must be added or to specify which pins must be isolated, use the create_isolation_rule command:

```
create_isolation_rule
    -name string
    [-isolation_condition expression | -no_condition]
    {-pins pin_list | -from power_domain_list | -to power_domain_list}...
    [-exclude pin_list]
    [-isolation_target {from|to}]
    [-isolation_output { high | low | hold | tristate}]
    [-secondary_domain power_domain]
```

Typically, isolation logic is needed to isolate signals going from a power domain being switched down to a power domain that remains on. However, if an input of a powered down domain requires a stable signal for electrical reasons, isolation is required even if the signal goes from a powered on domain to a powered down domain. Also, when a power domain is in the standby state, and all inputs to this power domain must be stable, isolation of the inputs will be required.

Referring to Table 2-5 on page 66, isolation logic is needed in power modes `slow` and `sleep` for any nets going from power domain `TDSPCore` to `AO` and `PLL`.

For the design in Figure 2-7 on page 64, one isolation rule was created.

The following command specifies to isolate all pins driving nets going from power domain `TDSPCore` to any other power domain. The pins must be isolated when the `iso_enable` signal becomes active low. The command further specifies that the output of the isolation gates must be `high` when the isolation condition is true.

```
create_isolation_rule -name ISORULE -from TDSPCore  \
    -isolation_condition "!PM_INST/iso_enable" -isolation_output high
```

Specifying the Rules to Create State Retention Logic

➤ To define the rule for replacing selected registers or all registers in the specified power domain with state retention registers, use the create_state_retention_rule command.

```
create_state_retention_rule
    -name string
    { -domain power_domain | -instances instance_list }
    [ -exclude instance_list ]
    [ -restore_edge expr | -save_edge expr
    | -restore_edge expr -save_edge expr
    | -restore_level expr -save_level expr]
    [ -restore_precondition expr] [-save_precondition expr]
    [-target_type {flop|latch|both}]
    [-secondary_domain domain]
```

For the design in Figure 2-7 on page 64, one rule was created. Because this design has only one power domain that is powered down, state retention rules are only needed for this power domain.

The following command creates a state retention rule for power domain `TDSPCore` and specifies that the states of the state retention cells in this domain will be saved when `pg_enable` signal becomes active high. The states of the state retention cells will be restored when the `pg_restore` signal becomes active low. The secondary power domain is not specified for the state retention logic. In this case, the tools will use the secondary (or base) power domain of its primary domain. That logic is the `PM_INST` instance which belongs to power domain `AO`.

```
create_state_retention_rule -name SRPG_TDSP \
    -domain TDSPCore \
    -restore_edge {!PM_INST/pg_restore} \
    -save_edge {PM_INST/pg_enable}
```

Specifying the Time to Transition between Power States

➤ To specify the transition states between power states for a power domain, use the update_power_domain command with one of the following options:

```
update_power_domain
    -name domain
    { -transition_slope [float:]float |
    | -transition_latency {from_nom latency_list}
    | -transition_cycles {from_nom cycle_list clock_pin} }
```

The design in Figure 2-7 on page 64 can have four possible transitions.

For power domain AO, the two possible transitions are from nominal condition low_ao to high_ao and vice versa.

For power domain TDSPCore, the two possible transitions are from nominal condition off to low_tdsp and vice versa. Typically, the transition time from the on state to the off state is not important and therefore it is not defined here.

The following command defines the minimum amd maximum transition time from nominal condition off to low_tdsp for power domain TDSPCore.

```
update_power_domain -name TDSPCore -transition_latency {off low_tdsp@2.0:2.5}
```

Specifying the Libraries to Use for a Condition

You already grouped libraries that are characterized for a specific set of operating conditions in a library set.

➤ To specify which library set to use for a specific nominal condition, use the update_nominal_condition command:

```
update_nominal_condition -name condition
    -library_set library_set
```

For the design in Figure 2-7 on page 64, library sets are specified for each nominal condition, except for nominal condition, off. The following command links library set ao_wc_0v99 to nominal condition high_ao.

```
update_nominal_condition -name high_ao -library_set ao_wc_0v99
```

Specifying the Power Constraints

Note: This information is optional in the CPF file.

➤ To specify the targets for leakage and dynamic power in the current design, use the `set_power_target` command:

```
set_power_target
    { -leakage float | -dynamic float
    | -leakage float -dynamic float}
```

For the design in Figure 2-7 on page 64, no power constraints were specified.

Specifying the Timing Constraints

Note: This information is optional in the CPF file.

➤ To specify the timing constraints for the current design, use the `-sdc_files` option of the `update_power_mode` command:

```
update_power_mode -name mode
    {-sdc_files | -setup_sdc_files | -hold_sdc_files} sdc_file_list
```

For the design in Figure 2-7 on page 64, separate timing constraints were defined for each of the modes.

The following command specifies the constraints to be used to optimize or analyze the design for the `full` power mode (when full functionality must be available). The directory of the SDC file is specified through a variable which was defined in the CPF file.

```
update_power_mode -name full \
    -sdc_files ${constraintDir}/dtmf_recvr_core_gate.sdc
```

Specifying the Activity Information

Note: This information is optional in the CPF file.

➤ To specify the activity information that can be used for power analysis, use

```
update_power_mode -name mode
    -activity_file file -activity_file_weight weight
```

Supported formats for the activity files are VCD, TCF, and SAIF.

If the design has several modes, you can specify the relative weight of the activities per mode.

For the design in Figure 2-7 `on page 64`, no activity information was specified.

Updating the Rules with Implementation Information

1. To specify the location or the type of level shifters to be used, use the update_level_shifter_rules command:

```
update_level_shifter_rules -names rule_list
    { -location {from | to | parent | any}
    | -within_hierarchy instance
    | -cells cell_list
    | -prefix string }...
```

   For the design in Figure 2-7 on page 64, three of the four level shifter rules were updated.

   The following command specifies that for rule LSRULE_H2L only level shifter cell LVLHLD2BWP can be used and all level shifters must be placed in the destination power domain.

```
update_level_shifter_rules -names LSRULE_H2L -cells LVLHLD2BWP -location to
```

2. To specify the location or the type of isolation cells to be used, use the update_isolation_rules command:

```
update_isolation_rules -names rule_list
    { -location {from | to | parent | any}
    | -within_hierarchy instance
    | -cells cell_list
    | -prefix string
    | -open_source_pins_only}...
```

   For the design in Figure 2-7 on page 64, only one isolation rule was created and this rule was updated.

   The following command specifies that for rule ISORULE only cell LVLLHCD2BWP can be used and all isolation cells must be placed in the destination power domain.

```
update_isolation_rules -names ISORULE -cells LVLLHCD2BWP -location to
```

3. To append the specified rules for state retention logic with implementation information, use the update_state_retention_rules command:

```
update_state_retention_rules
    -names rule_list
    { -cell_type string
    | -cells cell_list| -set_reset_control} ...
```

   For the design in Figure 2-7 on page 64, only one state retention rule was created and this rule was updated.

   The following command specifies that for rule SRPG_TDSP, only the following state retention cell RSDFCSRHD2BWP from library set tdsp_wc_0v792 can be used.

```
update_state_retention_rules -names SRPG_TDSP \
    -cell RSDFCSRHD2BWP -library_set tdsp_wc_0v792
```

Specifying the Global Power and Ground Nets

➤ To declare (or create) the nets connected to the ground and power supplies, use the create_ground_nets and create_power_nets commands:

```
create_ground_nets -nets net_list
    [-voltage string]
    [-external_shutoff_condition expression | -internal]
    [-user_attributes string_list]
    [-peak_ir_drop_limit float]
    [-average_ir_drop_limit float]

create_power_nets -nets net_list
    [-voltage string]
    [-external_shutoff_condition expression | -internal]
    [-user_attributes string_list]
    [-peak_ir_drop_limit float]
    [-average_ir_drop_limit float]
```

Note: Power and ground nets referenced in an `update_power_domain` command can be either always on or can be switchable power nets depending on the power domain specification. Other power and ground nets are considered to be always on unless you specify the `-external_shutoff_condition` option.

For the design in Figure 2-7 on page 64, four power nets and two ground nets are declared. One power net is connected to the variable power supply. One power net is needed for the `PLL` power domain. Two power nets are needed for the `TDSPCore` power domain: one power net can be switched off, another power net is needed to retain the states of the state retention cells.

The following command declares power net `VDD`. This net is connected to a power supply that can vary from 0.792 V to 0.99V in increments of 0.198V. This power net belongs to power domain `AO` because that is the power domain whose voltage can be dynamically scaled.

```
create_power_nets -nets VDD -voltage {0.792:0.99:0.198}
```

Specifying the Global Connections

➤ To specify how to connect global nets, such as power and ground nets to the cell pins, use the create_global_connection command:

```
create_global_connection
    -net net
    -pins pin_list
    [-domain domain | -instances instance_list]
```

If you omit the -domain or -instances option, the global connection applies to the specified pins of the entire design.

If you combine -pins and -domain options, only those pins in the specified list that also belong to the specified power domain are connected.

If you combine -pins and -instances options, only those pins in the specified list that also belong to the specified instances are connected.

For the design in Figure 2-7 on page 64, eleven global connections were specified.

The following command specifies that net VDD must be connected to all cell pins named VDD in power domain AO.

```
create_global_connection -domain AO -net VDD -pins VDD
```

Specifying the Rules for the Power Switch Logic

1. To specify how a single power switch must connect the external and internal power or ground nets for the specified power domain, use the create_power_switch_rule command.

```
create_power_switch_rule
    -name string
    -domain power_domain
    {-external_power_net net | -external_ground_net net}
```

   You can specify one or more commands for a power domain depending on whether you want to control the switchable power domain by a single switch or multiple switches.

2. To append the specified rules for power switch logic with implementation information, use the update_power_switch_rule command:

```
update_power_switch_rule
    -name string
    { -enable_condition_1 expression [-enable_condition_2 expression]
    | -acknowledge_receiver_1 express [-acknowledge_receiver_2 express]
    | -cells cell_list
    | -gate_bias_net power_net
    | -prefix string
    | -peak_ir_drop_limit float
    | -average_ir_drop_limit float } ...
```

For the design in Figure 2-7 on page 64, one rule was created. Because this design has only one power domain that is powered down, power switch rules are only needed for this power domain.

The first command creates a power switch rule for power domain TDSPCore and specifies that the source pin of the power switch must be connected to net VDD_TDSP_R.

The second command specifies that for rule TDSPCore_SW, only cell HDRDID1BWPHVT can be used. The command also specifies to use the CDN_SW_ prefix for the created logic and to connect the switch_en_out pin to the output pin of the power switch cell.

```
create_power_switch_rule -name TDSPCore_SW -domain TDSPCore \
    -external_power_net VDD_TDSP_R
update_power_switch_rule -name TDSPCore_SW -cells HDRDID1BWPHVT \
    -prefix CDN_SW_ -acknowledge_receiver_1 switch_en_out
```

Specifying Additional Information for Power and Ground Routing

➤ To specify additional information that applies to power and ground routing, use the update_power_domain command with one of the following options:

```
update_power_domain
    -name domain
    { -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
    | -user_attributes string_list} ...
```

For the design in Figure 2-7 on page 64, three update_power_domain commands were specified.

The following command specifies that AVDD is the main power net for all functional gates in power domain PLL, while Avss is the ground net.

```
update_power_domain -name PLL -primary_power_net Avdd -primary_ground_net Avss
```

Specifying the Operating Corners

The design must be able to perform under different sets of operating conditions (process, voltage, and temperature values). Because the timing and power characteristics of the cells depend on the operating conditions, different library sets that contain the characterization information for the different conditions are used. An operating corner shows which library set to use for a given operating condition.

Tip

Different portions of the design can operate at the same voltage and yet use dedicated library sets. In this case, it is possible to have multiple operating corners with the same operating conditions. Table 2-7 on page 69 illustrates this case.

➤ To define an operating corner, use the `create_operating_corner` command.

```
create_operating_corner
    -name string
    -voltage float [-ground_voltage float]
    [-process float]
    [-temperature float]
    -library_set library_set
```

Note: The `-pmos_bias_voltage` and `-nmos_bias_voltage` options of `create_operating_corner` are irrelevant for a design not using substrate biasing.

For the design in Figure 2-7 on page 64, six operating corners were specified.

The following command specifies to use library set `ao_wc_0v99` for operating corner `WCCOM_AO`. The process value of the operating corner is 1, the temperature is 125°C and the operating voltage is 0.99V. This operating corner is defined for worst case conditions.

```
create_operating_corner -name WCCOM_AO \
    -process 1 -temperature 125 -voltage 0.99 -library_set ao_wc_0v99
```

Specifying the Analysis Views

The design must function correctly in each power mode not only under typical conditions, but also under extreme conditions. Typically a multi-mode multi-corner timing analysis will be done for the worst case and the best case conditions.

An analysis view associates a specific operating corner with each power domain in the specified power mode. You need to make sure that all operating corners for a view correspond to either best or worst case conditions.

➤ To define an analysis view, use the `create_analysis_view` command.

```
create_analysis_view
    -name string
    -mode mode
    -domain_corners domain_corner_list
    [-user_attributes string_list]
```

Note: The `-group_views` option of `create_analysis_view` is only relevant in a hierarchical flow. For more information, refer to Chapter 4, “Hierarchical Flow.”

Use the following format to specify a domain corner:

domain_name@corner_name

Tip

The CPF file can contain several analysis views with the same domain and corner information for a power mode. For a multi-mode multi-corner analysis, some implementation and timing analysis tools need unique views to associate with different parasitic corners.

For the design in Figure 2-7 on page 64, eight analysis views were specified.

The following command specifies analysis view `AV_full_MIN_RC1` for power mode `full`. For this view the operating corners for the best case operating conditions are associated with the three power domains.

```
create_analysis_view -name AV_full_MIN_RC1 -mode full \
    -domain_corners {AO@BCCOM_AO PLL@BCCOM_AO TDSPCore@BC08COM_TDSP}
```

3

Process of Creating the CPF Content

Overview

The content of the CPF file can grow through the design process. The tools in the design process need different information. Therefore you can start the design with an incomplete CPF file.

- At the design stage, low power verification checks the following aspects of a design using PSO methodology:
  - Power up and power down of the design
  - State retention and restoration
  - Enabling and disabling of isolation

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

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

- At the logic implementation stage, logic synthesis adds the required level shifter logic, isolation logic, state retention logic, and power switch logic to the gate-level netlist, and test synthesis adds the required test logic.

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

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

- At the physical implementation stage, ATPG, and signoff, you need at least the following commands in addition to all commands listed above:

  ```
  create_power_nets
  create_ground_nets
  create_global_connection
  create_power_switch_rule
  update_power_switch_rule
  ```

```
update_power_domain
create_operating_corner
create_analysis_view
```

Note: You can run gate-level simulation after both logic and physical implementation.

Even when the CPF file that you read in contains more CPF commands than the tool needs, the tool will just read the ones it needs and will ignore the other ones.

The content change can be handled in several ways:

- You can start with a CPF file and add to this file as you go through the design process.

  See Using a Single CPF File.

- You can include additional commands in a separate file and source it in the original CPF file.

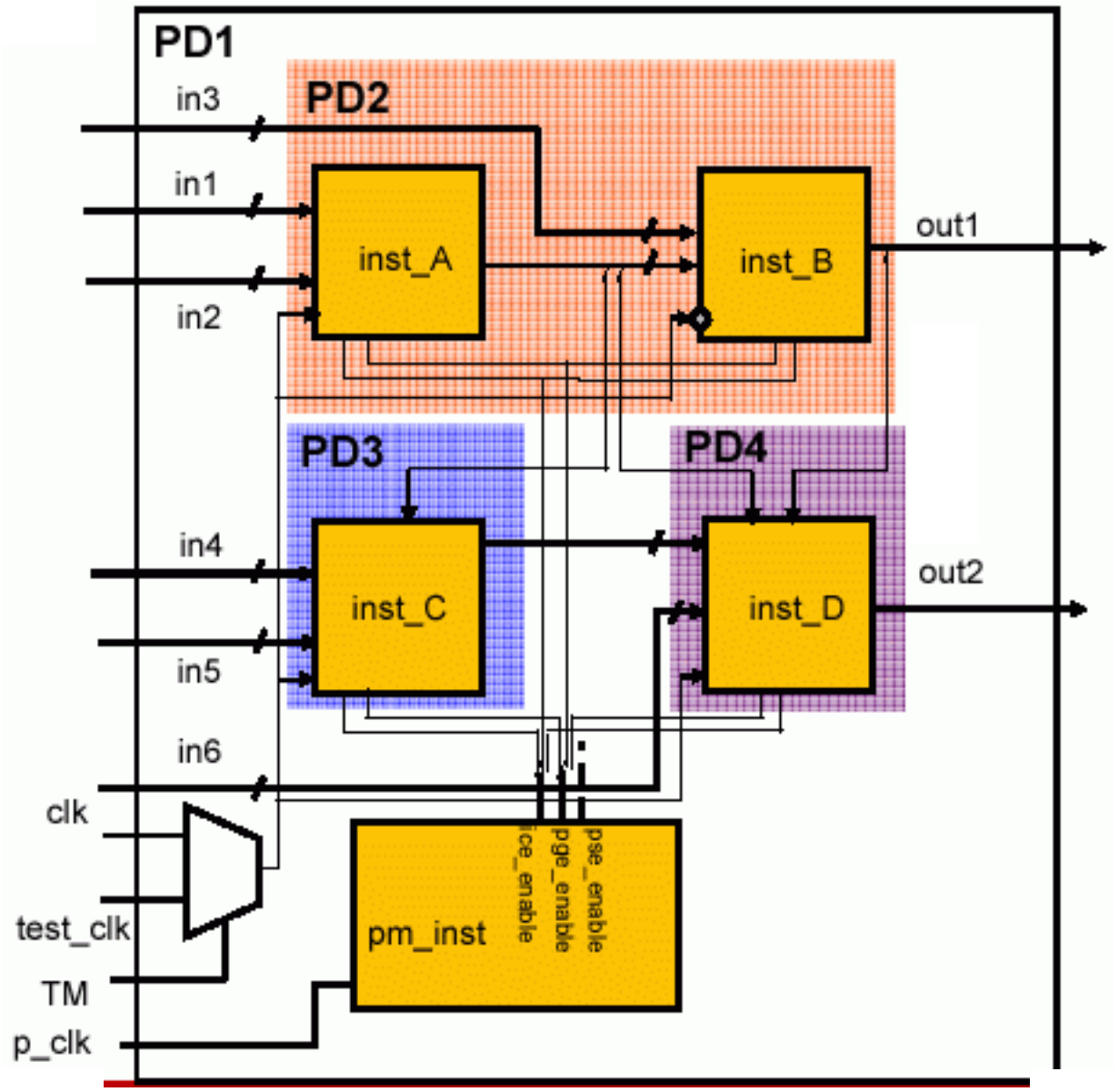
  See Using Multiple CPF Files.

A CPF file contains both the power intent for the design and the technology information. When you use multiple CPF files, you can capture the technology information in a separate file. Library-related definitions are captured in the `define_xxx` commands.

The power intent for the design captures the different power domains, the different power modes and transitions, and the specifications for the power logic. The power intent of the design is captured in the `create_xxx` commands. Implementation tools need more detailed information which is captured in the `update_xxx` commands. This information can be captured in a separate file.

The example shown in Figure 3-1 on page 104 is used to illustrate different approaches of CPF creation.

Figure 3-1 Example Design for CPF



Using a Single CPF File

```
################################################
#       Technology part of the CPF
################################################
```

 # define the level shifters

```
define_level_shifter_cell -cells LVLHLEHX* \
    -input_voltage_range 1.2 \
    -output_power_pin VDD \
    -output_voltage_range 0.8 \
    -direction down \
    -output_voltage_input_pin EN \
    -ground VSS \
    -enable EN \
    -valid_location to

define_level_shifter_cell -cells LVLHLELX* \
    -input_voltage_range 1.2 \
    -output_power_pin VDD \
    -output_voltage_range 0.8 \
    -direction down \
    -output_voltage_input_pin EN \
    -ground VSS \
    -enable EN \
    -valid_location to

define_level_shifter_cell -cells LVLHLX* \
    -input_voltage_range 1.2 \
    -output_power_pin VDD \
    -output_voltage_range 0.8 \
    -direction down \
    -ground VSS \
    -valid_location to

define_level_shifter_cell -cells LVLLHEHX* \
    -input_voltage_range 0.8 \
    -output_power_pin VDD \
    -input_power_pin VDDI \
    -output_voltage_range 1.2 \
    -direction up \
    -output_voltage_input_pin EN \
    -ground VSS \
    -valid_location to

define_level_shifter_cell -cells LVLLHX* \
    -input_voltage_range 0.8 \
    -output_voltage_range 1.2 \
    -input_power_pin VDDI \
    -output_power_pin VDD \
    -direction up \
    -ground VSS \
    -valid_location to
```

 # define the isolation cells

```
define_isolation_cell -cells ISOLN* \
    -enable EN \
    -power VDD -ground VSS \
    -valid_location to
```

```
# define the always on cell
define_always_on_cell -cells "BUFGX2M BUFGX8M INVGX2M INVGX8M" \
    -power_switchable VDD -power VDDG -ground VSS
# define the power switch cells
define_power_switch_cell -cells "HEAD8DM HEAD16DM HEAD32DM HEAD64DM" \
    -stage_1_enable !SLEEP \
    -type header \
    -stage_1_output SLEEPOUT \
    -power VDDG \
    -power_switchable VDD
define_power_switch_cell -cells "HEAD8M HEAD16M HEAD32M HEAD64M" \
    -stage_1_enable !SLEEP \
    -type header \
    -power VDDG \
    -power_switchable VDD
define_power_switch_cell -cells "FOOT8DM FOOT16DM" \
    -stage_1_enable SLEEPN \
    -type footer \
    -stage_1_output SLEEPNOUT \
    -ground VSSG \
    -ground_switchable VSS
define_power_switch_cell -cells "FOOT8M FOOT16M" \
    -stage_1_enable SLEEPN \
    -type footer \
    -ground VSSG \
    -ground_switchable VSS
# define the state retention cell
define_state_retention_cell -cells *DRFF* -restore_function RETN \
    -power VDDG \
    -power_switchable VDD \
    -ground VSS
# define the library sets
set vendor_08v_list "\
../LIBS/vendor_130/vendor13sp_tt_0p8v_25c.lib \
../LIBS/vendor_130LL/vendorl130e_ll_tt_0p8v_25c.lib \
../LIBS/vendor_130SP/vendorl130e_sp_tt_0p8v_25c.lib "
set vendor_120v_list "\
../LIBS/vendor_130/vendor13sp_tt_0p8v_1p2v_25c.lib \
../LIBS/vendor_130LL/vendorl130e_ll_tt_1p2v_25c.lib \
../LIBS/vendor_130SP/vendorl130e_sp_tt_1p2v_25c.lib "
define_library_set -name vendor_120v -libraries $vendor_120v_list
define_library_set -name vendor_08v -libraries $vendor_08v_list


################################################
#       Design part of the CPF
################################################
set_design top
set_hierarchy_separator "/"


# create power domains
create_power_domain -name PD1 -default
```

```
create_power_domain -name PD2 -instances {inst_A inst_B} \
    -shutoff_condition {pm_inst/pse_enable[0]}
create_power_domain -name PD3 -instances inst_C \
    -shutoff_condition {pm_inst/pse_enable[1]}
create_power_domain -name PD4 -instances inst_D \
    -shutoff_condition {pm_inst/pse_enable[2]}
create_power_domain -name LD1 -instances pm_inst/pd_inst*
# define the nominal conditions
create_nominal_condition -name point_8v -voltage 0.8
update_nominal_condition -name point_8v -library_set vendor_08v
create_nominal_condition -name one_p_2v -voltage 1.2
update_nominal_condition -name one_p_2v -library_set vendor_120v
create_nominal_condition -name off -voltage 0
# specify the power mode info
create_power_mode -name PM1 -default -domain_conditions {PD1@point_8v \
PD2@point_8v PD3@point_8v PD4@point_8v LD1@one_p_2v}
update_power_mode -name PM1 -sdc_files ../CPF/1.0/cm1.sdc
create_power_mode -name PM2 -domain_conditions {PD1@point_8v PD2@off \
PD3@point_8v PD4@point_8v LD1@one_p_2v}
create_power_mode -name PM3 -domain_conditions {PD1@point_8v PD2@off PD3@off \
PD4@point_8v LD1@one_p_2v}
create_power_mode -name PM4 -domain_conditions {PD1@point_8v PD2@off PD3@off \
PD4@off LD1@one_p_2v}
# create rules for state retention insertion
create_state_retention_rule -name st1 -domain PD2 \
-restore_edge {!pm_inst/pge_enable[0]}
create_state_retention_rule -name st2 -domain PD3 \
-restore_edge {!pm_inst/pge_enable[1]}
create_state_retention_rule -name st3 -domain PD4 \
-restore_edge {!pm_inst/pge_enable[2]}
# create rules for isolation logic insertion
create_isolation_rule -name iso1 -from PD2 \
-isolation_condition {pm_inst/ice_enable[0]}
create_isolation_rule -name iso2 -from PD3 \
-isolation_condition {pm_inst/ice_enable[1]}
create_isolation_rule -name iso3 -from PD4 \
-isolation_condition {pm_inst/ice_enable[2]}
# create rules for level shifter insertion
create_level_shifter_rule -name ls1 -to LD1
create_level_shifter_rule -name ls2 -from LD1
# declare power and ground nets
create_power_nets -nets VDD_0.8 -voltage 0.8
create_power_nets -nets VDD2 -voltage 0.8
create_power_nets -nets VDD3 -voltage 0.8
create_power_nets -nets VDD4 -voltage 0.8
create_power_nets -nets VDDH -voltage 1.2
create_ground_nets -nets VSS
# create global connections
```

```
create_global_connection -domain PD1 -net VDD_0.8 -pins VDD
create_global_connection -domain PD1 -net VDD_0.8 -pins VDDG
create_global_connection -domain PD1 -net VSS -pins VSS
create_global_connection -domain PD1 -net VSS -pins VSSG

create_global_connection -domain PD2 -net VDD2 -pins VDD
create_global_connection -domain PD2 -net VDD_0.8 -pins VDDG
create_global_connection -domain PD2 -net VSS -pins VSS
create_global_connection -domain PD2 -net VSS -pins VSSG

create_global_connection -domain PD3 -net VDD3 -pins VDD
create_global_connection -domain PD3 -net VDD_0.8 -pins VDDG
create_global_connection -domain PD3 -net VSS -pins VSS
create_global_connection -domain PD3 -net VSS -pins VSSG

create_global_connection -domain PD4 -net VDD4 -pins VDD
create_global_connection -domain PD4 -net VDD_0.8 -pins VDDG
create_global_connection -domain PD4 -net VSS -pins VSS
create_global_connection -domain PD4 -net VSS -pins VSSG

create_global_connection -domain LD1 -net VDDH -pins VDD
create_global_connection -domain LD1 -net VDD_0.8 -pins VDDI
create_global_connection -domain LD1 -net VSS -pins VSS
```

 # add implementation info for power domains

```
update_power_domain -name LD1 -internal_power_net VDDH
update_power_domain -name PD1 -internal_power_net VDD_0.8
update_power_domain -name PD2 -internal_power_net VDD2
update_power_domain -name PD3 -internal_power_net VDD3
update_power_domain -name PD4 -internal_power_net VDD4
```

 # update the rules

```
update_state_retention_rules -names st1 -cell_type DRFF -library_set vendor_08v
update_state_retention_rules -names st2 -cell_type DRFF -library_set vendor_08v
update_state_retention_rules -names st3 -cell_type DRFF -library_set vendor_08v

update_isolation_rules -names iso1 -location to -cells ISOLNX2M
update_isolation_rules -names iso2 -location to -cells ISOLNX2M
update_isolation_rules -names iso3 -location to -cells ISOLNX2M

update_level_shifter_rules -names ls1 -cells LVLLHX2M -location to -prefix RC_LS
```

 # specify the power switch rule information

```
create_power_switch_rule -name SW1 -domain PD2 -external_power_net VDD_0.8
update_power_switch_rule -name SW1 -cells HEAD32M -prefix CDN_

create_power_switch_rule -name SW2 -domain PD3 -external_power_net VDD_0.8
update_power_switch_rule -name SW2 -cells HEAD32M -prefix CDN_

create_power_switch_rule -name SW3 -domain PD4 -external_power_net VDD_0.8
update_power_switch_rule -name SW3 -cells HEAD32M -prefix CDN_

end_design
```

Using Multiple CPF Files

In this case a master CPF file is created in which two other supporting CPF files with additional CPF commands are sourced.

- `pf_design_1.cpf` (see Figure 3-2 on page 110) contains the basic information used by most tools.
- `library_vendor_130.cpf` (see Figure 3-3 on page 112) contains the technology-related information.

  This information will typically be provided by the library vendor.
- `pf_design_1_impl.cpf` (see Figure 3-4 on page 114) contains implementation-specific information.

  The commands in this file can be added later in the flow.

Figure 3-2 Content of the pf_design_1.cpf File (Master CPF File)

```
#start of master CPF file
source library_vendor_130.cpf
set vendor_08v_list "\
../LIBS/vendor_130/vendor13sp_tt_0p8v_25c.lib \
../LIBS/vendor_130LL/vendorl130e_ll_tt_0p8v_25c.lib \
../LIBS/vendor_130SP/vendorl130e_sp_tt_0p8v_25c.lib "
set vendor_120v_list "\
../LIBS/vendor_130/vendor13sp_tt_0p8v_1p2v_25c.lib \
../LIBS/vendor_130LL/vendorl130e_ll_tt_1p2v_25c.lib \
../LIBS/vendor_130SP/vendorl130e_sp_tt_1p2v_25c.lib "
define_library_set -name vendor_120v -libraries $vendor_120v_list
define_library_set -name vendor_08v -libraries $vendor_08v_list


################################################
#       Design part of the CPF
################################################
set_design top
set_hierarchy_separator "/"
# create power domains
create_power_domain -name PD1 -default
create_power_domain -name PD2 -instances {inst_A inst_B} \
    -shutoff_condition {pm_inst/pse_enable[0]}
create_power_domain -name PD3 -instances inst_C \
    -shutoff_condition {pm_inst/pse_enable[1]}
create_power_domain -name PD4 -instances inst_D \
    -shutoff_condition {pm_inst/pse_enable[2]}
create_power_domain -name LD1 -instances pm_inst/pd_inst*
# create nominal conditions
create_nominal_condition -name point_8v -voltage 0.8
create_nominal_condition -name one_p_2v -voltage 1.2
create_nominal_condition -name off -voltage 0
# create power modes
create_power_mode -name PM1 -default -domain_conditions {PD1@point_8v \
PD2@point_8v PD3@point_8v PD4@point_8v LD1@one_p_2v}
create_power_mode -name PM2 -domain_conditions {PD1@point_8v PD2@off \
PD3@point_8v PD4@point_8v LD1@one_p_2v}
create_power_mode -name PM3 -domain_conditions {PD1@point_8v PD2@off \
PD3@off PD4@point_8v LD1@one_p_2v}
create_power_mode -name PM4 -domain_conditions {PD1@point_8v PD2@off \
PD3@off PD4@off LD1@one_p_2v}
# create rules for state retention insertion
create_state_retention_rule -name st1 -domain PD2 \
-restore_edge {!pm_inst/pge_enable[0]}
create_state_retention_rule -name st2 -domain PD3 \
-restore_edge {!pm_inst/pge_enable[1]}
create_state_retention_rule -name st3 -domain PD4 \
-restore_edge {!pm_inst/pge_enable[2]}
```

```
# create rule for isolation logic insertion
create_isolation_rule -name iso1 -from PD2 \
-isolation_condition {pm_inst/ice_enable[0]}
create_isolation_rule -name iso2 -from PD3 \
-isolation_condition {pm_inst/ice_enable[1]}
create_isolation_rule -name iso3 -from PD4 \
-isolation_condition {pm_inst/ice_enable[2]}
# create rules for level shifter insertion
create_level_shifter_rule -name ls1 -to LD1
create_level_shifter_rule -name ls2 -from LD1
source pf_design_1_impl.cpf
end_design
```

Figure 3-3 Content of the library_vendor_130.cpf File

```
################################################
#       Technology part of the CPF
################################################
# define the level shifters
define_level_shifter_cell -cells LVLHLEHX* \
    -input_voltage_range 1.2 \
    -output_power_pin VDD \
    -output_voltage_range 0.8 \
    -direction down \
    -output_voltage_input_pin EN \
    -ground VSS \
    -enable EN \
    -valid_location to
define_level_shifter_cell -cells LVLHLELX* \
    -input_voltage_range 1.2 \
    -output_power_pin VDD \
    -output_voltage_range 0.8 \
    -direction down \
    -output_voltage_input_pin EN \
    -ground VSS \
    -enable EN \
    -valid_location to
define_level_shifter_cell -cells LVLHLX* \
    -input_voltage_range 1.2 \
    -output_power_pin VDD \
    -output_voltage_range 0.8 \
    -direction down \
    -ground VSS \
    -valid_location to
define_level_shifter_cell -cells LVLLHEHX* \
    -input_voltage_range 0.8 \
    -output_power_pin VDD \
    -input_power_pin VDDI \
    -output_voltage_range 1.2 \
    -direction up \
    -output_voltage_input_pin EN \
    -ground VSS \
    -valid_location to
define_level_shifter_cell -cells LVLLHX* \
    -input_voltage_range 0.8 \
    -output_voltage_range 1.2 \
    -input_power_pin VDDI \
    -output_power_pin VDD \
    -direction up \
    -ground VSS \
    -valid_location to
# define the isolation cells
define_isolation_cell -cells ISOLN* \
    -enable EN \
    -power VDD -ground VSS \
    -valid_location to
# define the always on cell
define_always_on_cell -cells "BUFGX2M BUFGX8M INVGX2M INVGX8M" \
    -power_switchable VDD -power VDDG -ground VSS
```

```
# define the power switch cells
define_power_switch_cell -cells "HEAD8DM HEAD16DM HEAD32DM HEAD64DM" \
    -stage_1_enable !SLEEP \
    -type header \
    -stage_1_output SLEEPOUT \
    -power VDDG \
    -power_switchable VDD
define_power_switch_cell -cells "HEAD8M HEAD16M HEAD32M HEAD64M" \
    -stage_1_enable !SLEEP \
    -type header \
    -power VDDG \
    -power_switchable VDD
define_power_switch_cell -cells "FOOT8DM FOOT16DM" \
    -stage_1_enable SLEEPN \
    -type footer \
    -stage_1_output SLEEPNOUT \
    -ground VSSG \
    -ground_switchable VSS
define_power_switch_cell -cells "FOOT8M FOOT16M" \
    -stage_1_enable SLEEPN \
    -type footer \
    -ground VSSG \
    -ground_switchable VSS
# define the state retention cell
define_state_retention_cell -cells *DRFF* -restore_function RETN \
    -power VDDG \
    -power_switchable VDD \
    -ground VSS
```

Figure 3-4 Content of the pf_design_1_impl.cpf File

```
#start of implementation information
# associate library sets with nominal conditions
update_nominal_condition -name point_8v -library_set vendor_08v
update_nominal_condition -name one_p_2v -library_set vendor_120v
# specify timing constraints
update_power_mode -name PM1 -sdc_files ../CPF/1.0/cm1.sdc
# declare power and ground nets
create_power_nets -nets VDD_0.8 -voltage 0.8
create_power_nets -nets VDD2 -voltage 0.8
create_power_nets -nets VDD3 -voltage 0.8
create_power_nets -nets VDD4 -voltage 0.8
create_power_nets -nets VDDH -voltage 1.2
create_ground_nets -nets VSS
# create global connections
create_global_connection -domain PD1 -net VDD_0.8 -pins VDD
create_global_connection -domain PD1 -net VDD_0.8 -pins VDDG
create_global_connection -domain PD1 -net VSS -pins VSS
create_global_connection -domain PD1 -net VSS -pins VSSG
create_global_connection -domain PD2 -net VDD2 -pins VDD
create_global_connection -domain PD2 -net VDD_0.8 -pins VDDG
create_global_connection -domain PD2 -net VSS -pins VSS
create_global_connection -domain PD2 -net VSS -pins VSSG
create_global_connection -domain PD3 -net VDD3 -pins VDD
create_global_connection -domain PD3 -net VDD_0.8 -pins VDDG
create_global_connection -domain PD3 -net VSS -pins VSS
create_global_connection -domain PD3 -net VSS -pins VSSG
create_global_connection -domain PD4 -net VDD4 -pins VDD
create_global_connection -domain PD4 -net VDD_0.8 -pins VDDG
create_global_connection -domain PD4 -net VSS -pins VSS
create_global_connection -domain PD4 -net VSS -pins VSSG
create_global_connection -domain LD1 -net VDDH -pins VDD
create_global_connection -domain LD1 -net VDD_0.8 -pins VDDI
create_global_connection -domain LD1 -net VSS -pins VSS
# add implementation info for power domains
update_power_domain -name LD1 -internal_power_net VDDH
update_power_domain -name PD1 -internal_power_net VDD_0.8
update_power_domain -name PD2 -internal_power_net VDD2
update_power_domain -name PD3 -internal_power_net VDD3
update_power_domain -name PD4 -internal_power_net VDD4
# update the rules
update_state_retention_rules -names st1 -cell_type DRFF -library_set vendor_08v
update_state_retention_rules -names st2 -cell_type DRFF -library_set vendor_08v
update_state_retention_rules -names st3 -cell_type DRFF -library_set vendor_08v
update_isolation_rules -names iso1 -location to -cells ISOLNX2M
update_isolation_rules -names iso2 -location to -cells ISOLNX2M
update_isolation_rules -names iso3 -location to -cells ISOLNX2M
update_level_shifter_rules -names ls1 -cells LVLLHX2M -location to -prefix RC_LS
```

```
# specify the rules for power switch insertion
create_power_switch_rule -name SW1 -domain PD2 -external_power_net VDD_0.8
update_power_switch_rule -name SW1 -cells HEAD32M -prefix CDN_
create_power_switch_rule -name SW2 -domain PD3 -external_power_net VDD_0.8
update_power_switch_rule -name SW2 -cells HEAD32M -prefix CDN_
create_power_switch_rule -name SW3 -domain PD4 -external_power_net VDD_0.8
update_power_switch_rule -name SW3 -cells HEAD32M -prefix CDN_
```

4

Hierarchical Flow

Introduction to Hierarchical Flow

In Chapter 2, "Creating a CPF File," you learned about the CPF constructs to describe three of the most used low power techniques in a non-hierarchical flow. In this chapter, you will learn more about the additional constructs you need to use for a bottom-up hierarchical flow.

Typically in a bottom-up *hierarchical* flow, the design instantiates IPs.

If an IP is designed or implemented with a complex power structure, the IP will have a separate CPF file to describe its power intent.

CPF distinguishes the following categories:

- A ***hard non-custom IP*** is a block that has been synthesized and placed and routed, but small modifications can still be made. It is also referred to as a *hard IP*.
- A ***hard custom IP*** is a block that is mostly implemented by hand. Custom IP users cannot make any changes to the block. Examples are third-party memories. A hard custom IP is also referred to as a *macro cell*.
- A ***soft IP*** is a design module that can be supplied by a third-party. Users have the full capability to (re-)synthesize and place and route the design. Examples are synthesizable CPU cores, and so on.

When the IP is instantiated at the top level, ***power domain mapping*** is required to indicate which power domains from the top-level need to provide power supplies to the IP.

Mapping a power domain of a block (IP) to another power domain at the top level when the block is instantiated in the top-level design involves

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

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

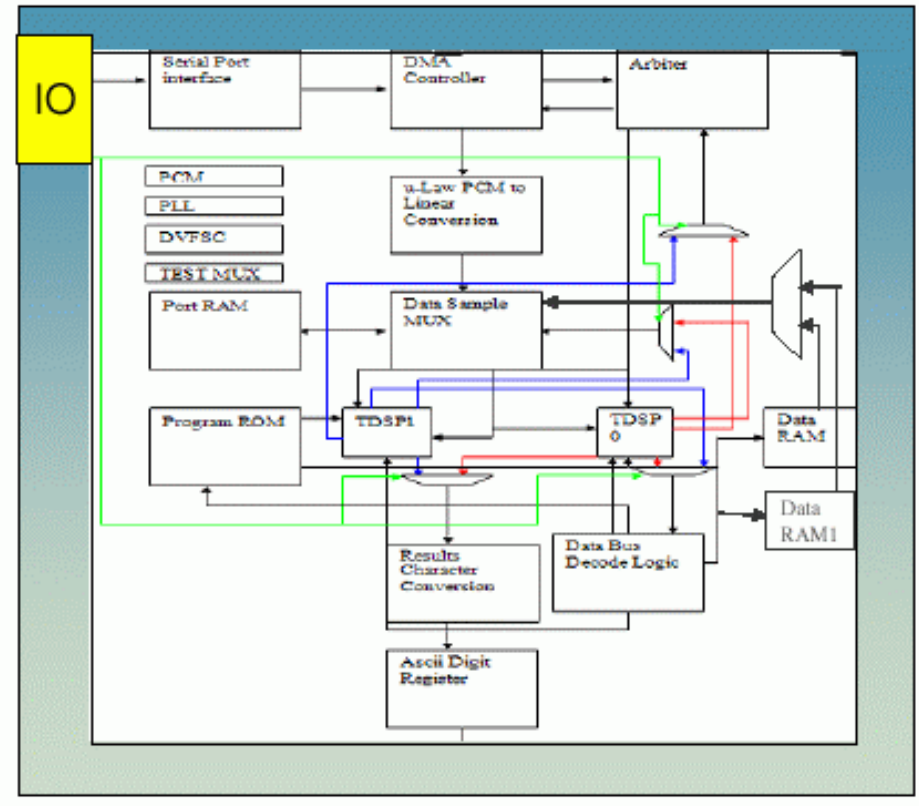
A simplified view of a DTMF receiver, the design used to illustrate the hierarchical flow, is shown in Figure 4-1 on page 119.

The DTMF receiver design uncompresses and accumulates an eight-bit uLaw encoded dual tone data sampled over a100 millisecond period into a dual buffer memory.

Once an entire sample has been received embedded DSP (TDSP) cores will read the input buffer and execute a modified discrete Fourier transform (DFT) algorithm to calculate a partial frequency spectrum. Upon completion the DSP cores forward the resulting frequency response data for comparison to values corresponding to DTMF digits. If a match is found the 8 bit ASCII digit is presented at the output of the core.

The voltage of the core is scaled depending on the requested function of the design. If the processing speed is critical a higher voltage is used, if the processing speed is not critical, the voltage is dynamically scaled down together with the clock frequency to save power. For this design we assume that the voltage supply is dynamically controlled external to the chip.

Figure 4-1 Example of Hierarchical Design



The core of the design is the `DTMF_INST` instance. The I/O pads are instantiated in a hierarchical instance `IOPADS_INST`. The DTMF receiver design further contains

- A PLL block

  This block is used to generate the clocks needed by all the blocks in the design. It has a reference clock, `refclk`, that is used to generate all other clocks.

  Because the design uses two operating voltages, two clock signals are created.

  This analog block needs to operate at a constant voltage to ensure correct operation. It therefore needs a dedicated power input, `Avdd`.

- Two TDSP cores: `TDSP0` and `TDSP1` (in Figure 4-1 on page 119)

  These two digital signal processing blocks can operate at a higher voltage and frequency, and at a lower voltage and frequency when the processing speed is not critical. When the blocks do not need to be operational, they are powered down.

  A `dvfs_en` signal at the input of the chip indicates to the `dvfs_controller` when the voltage and frequency can be adjusted. The controller sends a signal to the PLL and to the external voltage regulator to adjust the clock frequency and the voltage, respectively.

  These blocks are instances of the soft block, `tdsp_core`. The `tdsp.cpf` file describes the power intent of this soft block.

- `Data RAM1` is a special RAM that can retain its memory when it is powered down. This RAM can operate at the same voltages as the design core.

  This RAM is an instance of the hard custom IP, `ram_256x16A`. The `ram.cpf` file describes the power intent of this RAM instance.

- `Data RAM` is a regular RAM.

  This RAM can only operate at one voltage. It can also be powered down when not operational. This RAM does not retain its memory when it is powered down.

- The `pm_inst` block (not shown in Figure 4-1 on page 119)

  This block generates all power control signals for the chip.

  This block operates at the same voltage as the top-level of the design.

The top-level of the design has the following power domains:

- The `PDpll` power domain contains the PLL block (`DTMF_INST/PLLCLK_INST`) and some of the IO pads (for example, `IOPADS_INST/Prefclkip`) that are connected to this block.

- The `PDtdsp` power domain contains the two TDSP cores (`DTMF_INST/TDSP_CORE_INST0, DTMF_INST/TDSP_CORE_INST1`) and the special RAM (`DTMF_INST/RAM_128x16_TEST_INST1`).
- The `PDram` power domain contains the regular RAM (`DTMF_INST/RAM_128x16_TEST_INST`).
- The `PDshutoff_io` power domain contains two I/O instances that can be powered down.
- The `PDdefault` power domain contains all other blocks that do not belong to any special power domain. All these blocks operate in DVFS mode.
- The `PDram_virtual` power domain contains no blocks.

Power domains `PDdefault`, `PDpll` and `PDram_virtual` are unswitched domains. Power domains `PDtdsp`, `PDram`, and `PDshutoff_io` are switchable domains.

Power domain `PDshutoff_io` is an on-chip controlled external switchable domain, while power domains `PDtdsp` and `PDram` are internal switchable domains; they can be switched off through an internal power switch network. The external power supply for the power switch network of power domain `PDtdsp` is provided by the power domain `PDdefault`, while the external power supply for the power switch network of power domain `PDram` is provided by the power domain `PDram_virtual`.`PDdefault` is the secondary power domain for domain `PDtdsp`, while `PDram_virtual` is the secondary power domain for domain `PDram`.

Table 4-1 shows the operating voltages for each of the power domains in the different power modes of the design.

Table 4-1 Voltages of the Power Domains at the Top Level in the Different Power Modes

Power Mode	Corresponding Power Domain					
	PDdefault	PDshutoff_io	PDpll	PDram_virtual	PDram	PDtdsp
PMdvfs1	0.81	0.81	0.81	0.72	0.72	0.81
PMdvfs1_off	0.81	0.81	0.81	0.72	0	0
PMdvfs1_shutoffio_off	0.81	0	0.81	0.72	0	0
PMdvfs2	0.72	0.72	0.81	0.72	0.72	0.72
PMdvfs2_off	0.72	0.81	0.81	0.72	0	0
PMdvfs2_shutoffio_off	0.72	0	0.81	0.72	0	0
PMscan	0.81	0.81	0.81	0.72	0.72	0.81

Table 4-2 on page 122 shows the different control signals needed at the top-level of the design to isolate logic, turn off the power supply switches, and save and restore the retention states. The always on domains do not need any control signals.

Table 4-2 Signals Controlling the Power Domains at the Top Level

Power Domain	Control Signals		
	power switch	isolation cell	state retention cell
PDdefault	no control signal	no control signal	no control signal
PDpll	no control signal	no control signal	no control signal
PDram_virtual	no control signal	no control signal	no control signal
PDram	!power_switch_enable	!isolation_enable	no control signal
PDtdsp	!power_switch_enable	!isolation_enable	no control signal[1]
PDshutoff_io	io_shutoff_ack	!spi_ip_isolate	no control signal

1. Although no state retention rules are specified at the top-level domain, state retention rules are defined for the special RAM, defined through the ram.cpf (see Macro CPF — ram.cpf), and for the soft block, defined through the tdsp.cpf (see Soft IP CPF — tdsp.cpf).

All control signals are generated by the power control manager (`PM_INST`) which is an instance of the core.

Table 4-3 on page 122 lists the operating corners for the design for the corresponding operating voltages.

Table 4-3 Operating Corners

Operating Corner	Operating Voltage
`PMdvfs2_bc`	`0.88`
`PMdvfs1_bc`	`0.99`
`PMdvfs1_wc`	`0.81`
`PMdvfs2_wc`	`0.72`

CPF for Hierarchical Flow lists the complete CPF files used for this design.

Steps to Create the CPF File describes the specifics to create a CPF file for a hierarchical flow, assuming your design is using a hard and soft IP for which you have the CPF descriptions.

CPF for Hierarchical Flow

Technology CPF File — tech.cpf

```
set_cpf_version 1.1
if { [set_instance] == [set_hierarchy_separator] } {
set libdir ../LIBS/N45GS/timing
set libdir_io ../LIBS/N65LP/timing
########################################################################
#
#       Technology part of the CPF
#
########################################################################
define_always_on_cell -cells PTBUFFD2BWP \
    -power_switchable VDD -power TVDD -ground VSS
define_isolation_cell -cells ISO* \
    -power VDD -ground VSS \
    -enable ISO \
    -valid_location to
define_level_shifter_cell -cells LVL*HLD* \
    -input_voltage_range 0.72:0.81:0.09 \
    -output_voltage_range 0.72:0.81:0.09 \
    -direction down \
    -output_power_pin VDD \
    -ground VSS \
    -valid_location to
define_level_shifter_cell -cells PTLVL*HLD* \
    -input_voltage_range 0.72:0.81:0.09 \
    -output_voltage_range 0.72:0.81:0.09 \
    -direction down \
    -output_power_pin TVDD \
    -ground VSS \
    -valid_location to
define_level_shifter_cell -cells LVLLHCD* \
    -input_voltage_range 0.72:0.81:0.09 \
    -output_voltage_range 0.72:0.81:0.09 \
    -enable NSLEEP \
    -input_power_pin VDDL \
    -output_power_pin VDD \
    -direction up \
    -ground VSS \
    -enable NSLEEP \
    -valid_location to
define_level_shifter_cell -cells LVLLHD* \
    -input_voltage_range 0.72:0.81:0.09 \
    -output_voltage_range 0.72:0.81:0.09 \
    -input_power_pin VDDL \
    -output_power_pin VDD \
    -direction up \
    -ground VSS \
    -valid_location to
define_power_switch_cell -cells {HDRDID1BWPHVT HDRDIAOND1BWPHVT} \
    -power_switchable VDD -power TVDD \
    -stage_1_enable NSLEEPIN1 -stage_1_output NSLEEPOUT1 \
    -stage_2_enable NSLEEPIN2 -stage_2_output NSLEEPOUT2 \
    -type header
```

```
define_state_retention_cell \
    -cells {MSRSDFCSNQHD2BWP} \
    -cell_type "master_slave" \
    -clock_pin CP \
    -power TVDD -power_switchable VDD \
    -ground VSS \
    -restore_check "!CP" \
    -save_function "!CP" \
    -always_on_components DFF_inst
define_state_retention_cell \
    -cells {RSDFCSRHD2BWP} \
    -clock_pin CP \
    -power TVDD -power_switchable VDD \
    -ground VSS \
    -save_function "SAVE" \
    -cell_type "ballon_latch" \
    -restore_function "!NRESTORE" \
    -always_on_components save_data


##########################################################################
## DEFINE LIBRARY SET
##########################################################################
set lib_0v81_wc_base "$libdir/tcbn45gsbwpwc.lib"
set lib_0v81_wc_extra "\
    $libdir/tcbn45lpbwp_c070208wc0d720d9_modified.lib \
    $libdir/tcbn45lpbwp_c070208wc0d90d9_modified.lib \
    $libdir/tcbn45lpbwp_c070208wc_modified.lib \
    $libdir/pllclk_slow.lib \
    $libdir/ram_256x16A_slow_syn.lib \
    $libdir/rom_512x16A_slow_syn.lib "
set lib_0v81_bc_base "$libdir/tcbn45gsbwpbc.lib "
set lib_0v81_bc_extra "\
    $libdir/tcbn45lpbwp_c070208bc0d881d1_modified.lib \
    $libdir/tcbn45lpbwp_c070208bc1d11d1_modified.lib \
    $libdir/tcbn45lpbwp_c070208bc_modified.lib \
    $libdir/pllclk_slow.lib \
    $libdir/ram_256x16A_slow_syn.lib \
    $libdir/rom_512x16A_slow_syn.lib "
set lib_0v72_wc_base "$libdir/tcbn45gsbwpwc_0d72.lib "
set lib_0v72_wc_extra "\
    $libdir/tcbn45lpbwp_c070208wc0d90d72_modified.lib \
    $libdir/tcbn45lpbwphvt_c070208wc0d72_modified.lib \
    $libdir/tcbn45lpbwp_c070208wc0d72_ptlvl_modified.lib \
    $libdir/tcbn45lpbwp_c070208wc0d720d72_modified.lib \
    $libdir/ram_256x16A_slow_syn.lib \
    $libdir/rom_512x16A_slow_syn.lib "
set lib_0v72_bc_base "$libdir/tcbn45gsbwpbc_0d88.lib "
set lib_0v72_bc_extra "\
    $libdir/tcbn45lpbwp_c070208bc1d10d88_modified.lib \
    $libdir/tcbn45lpbwphvt_c070208bc0d88_modified.lib \
    $libdir/tcbn45lpbwp_c070208bc0d88_ptlvl_modified.lib \
    $libdir/tcbn45lpbwp_c070208bc0d880d88_modified.lib \
    $libdir/ram_256x16A_fast_syn.lib \
    $libdir/rom_512x16A_slow_syn.lib "
```

```
set io_lib_wc "$libdir_io/tpzn65lpgv2wc.lib"
set io_lib_bc "$libdir_io/tpzn65lpgv2bc.lib"
define_library_set -name wc_0v81 -libraries "$lib_0v81_wc_base \
    $lib_0v81_wc_extra $io_lib_wc"
define_library_set -name bc_0v81 -libraries "$lib_0v81_bc_base \
    $lib_0v81_bc_extra $io_lib_bc"
define_library_set -name wc_0v72 -libraries "$lib_0v72_wc_base \
    $lib_0v72_wc_extra $io_lib_wc"
define_library_set -name bc_0v72 -libraries "$lib_0v72_bc_base \
    $lib_0v72_bc_extra $io_lib_wc"
}
```

Top CPF File

```
set_cpf_version 1.1
set_hierarchy_separator /
set_design dtmf_chip
set constraintDir $env(test_PATH)/RELEASE/mmmc
include tech.cpf
################################################################################
## create power domains
################################################################################
create_power_domain -name PDdefault -default
update_power_domain -name PDdefault -primary_power_net VDD -primary_ground_net VSS

create_power_domain -name PDshutoff_io \
    -instances {IOPADS_INST/Pspifsip IOPADS_INST/Pspidip} \
    -boundary_ports {spi_fs spi_data} \
    -shutoff_condition {io_shutoff_ack} \
    -external_controlled_shutoff
update_power_domain -name PDshutoff_io -primary_power_net VDD_IO \
    -primary_ground_net VSS
create_power_domain -name PDpll  \
    -instances {DTMF_INST/PLLCLK_INST \
                IOPADS_INST/Pibiasip \
                IOPADS_INST/Ppllrstip \
                IOPADS_INST/Prefclkip \
                IOPADS_INST/Pvcomop \
                IOPADS_INST/Pvcopop} \
    -boundary_ports { ibias reset refclk vcom vcop pllrst }
update_power_domain -name PDpll -primary_power_net Avdd -primary_ground_net Avss
create_power_domain -name PDram_virtual
update_power_domain -name PDram_virtual -primary_power_net VDDL \
    -primary_ground_net VSS
create_power_domain -name PDram \
    -shutoff_condition {!DTMF_INST/PM_INST/power_switch_enable} \
    -base_domains PDram_virtual \
    -instances  DTMF_INST/RAM_128x16_TEST_INST
update_power_domain -name PDram -primary_power_net VDDL_sw -primary_ground_net VSS
create_power_domain -name PDtdsp \
    -shutoff_condition {!DTMF_INST/PM_INST/power_switch_enable} \
    -base_domains PDdefault
update_power_domain -name PDtdsp -primary_power_net VDD_sw -primary_ground_net VSS
################################################################################
## create power modes
################################################################################
create_nominal_condition -name nom_0v81 -voltage 0.81
update_nominal_condition -name nom_0v81 -library_set wc_0v81
create_nominal_condition -name nom_0v72 -voltage 0.72
update_nominal_condition -name nom_0v72 -library_set wc_0v72
create_power_mode -name PMdvfs1 \
    -domain_conditions {PDpll@nom_0v81 PDdefault@nom_0v81 PDtdsp@nom_0v81 \
```

```
                        PDram@nom_0v72 PDshutoff_io@nom_0v81 \
                        PDram_virtual@nom_0v72} \
    -default
update_power_mode -name PMdvfs1 -sdc_files ${constraintDir}/dtmf_dvfs1.sdc


create_power_mode -name PMdvfs1_off \
    -domain_conditions {PDpll@nom_0v81 PDdefault@nom_0v81 PDshutoff_io@nom_0v81 \
                        PDram_virtual@nom_0v72}


create_power_mode -name PMdvfs1_shutoffio_off \
    -domain_conditions {PDpll@nom_0v81 PDdefault@nom_0v81 PDram_virtual@nom_0v72}


create_power_mode -name PMdvfs2 \
    -domain_conditions {PDpll@nom_0v81 PDdefault@nom_0v72 PDtdsp@nom_0v72 \
                        PDram@nom_0v72 PDshutoff_io@nom_0v72 \
                        PDram_virtual@nom_0v72}
update_power_mode -name PMdvfs2 -sdc_files ${constraintDir}/dtmf_dvfs2.sdc


create_power_mode -name PMdvfs2_off  \
    -domain_conditions {PDpll@nom_0v81 PDdefault@nom_0v72 PDshutoff_io@nom_0v72  \
                        PDram_virtual@nom_0v72}


create_power_mode -name PMdvfs2_shutoffio_off \
    -domain_conditions {PDpll@nom_0v81 PDdefault@nom_0v72  PDram_virtual@nom_0v72}


create_power_mode -name PMscan  \
    -domain_conditions {PDpll@nom_0v81 PDdefault@nom_0v81 PDtdsp@nom_0v81 \
    PDram@nom_0v72 PDshutoff_io@nom_0v81  PDram_virtual@nom_0v72}
update_power_mode -name PMscan -sdc_files ${constraintDir}/dtmf_scan.sdc


########################################################################
## domain mapping
########################################################################
set_instance DTMF_INST/TDSP_CORE_INST0 \
    -domain_mapping { {PDtdsp_block PDtdsp} } \
    -port_mapping { {retention_save DTMF_INST/PM_INST/state_retention_save} \
                {retention_restore DTMF_INST/PM_INST/state_retention_restore} }
include tdsp.cpf
set_instance DTMF_INST/TDSP_CORE_INST1 \
    -domain_mapping { {PDtdsp_block PDtdsp} } \
    -port_mapping { {retention_save DTMF_INST/PM_INST/state_retention_save} \
                {retention_restore DTMF_INST/PM_INST/state_retention_restore} }
include tdsp.cpf
set_instance DTMF_INST/RAM_128x16_TEST_INST1/RAM_128x16_INST \
    -domain_mapping { {RAM_DEFAULT PDtdsp} }
include ram.cpf
```

```
###########################################################################
## create isolation and level shifter rules
###########################################################################
create_isolation_rule -name ISORULE1 \
    -from PDtdsp \
    -to PDdefault \
    -isolation_condition {!DTMF_INST/PM_INST/isolation_enable} \
    -isolation_output high
create_isolation_rule -name ISORULE3 \
    -from PDram \
    -to PDdefault \
    -isolation_condition {!DTMF_INST/PM_INST/isolation_enable} \
    -isolation_output high
create_isolation_rule -name ISORULE4  \
    -from PDshutoff_io \
    -isolation_condition {!DTMF_INST/PM_INST/spi_ip_isolate} \
    -isolation_output low


create_level_shifter_rule -name LSRULE_H2L3 \
    -from PDdefault \
    -to  PDram \
    -exclude { DTMF_INST/PM_INST/power_switch_enable }
update_level_shifter_rules -names LSRULE_H2L3 -location to -cells LVLHLD2BWP


create_level_shifter_rule -name LSRULE_H2L3_SW \
    -from PDdefault \
    -to  PDram \
    -pins { DTMF_INST/PM_INST/power_switch_enable }
update_level_shifter_rules -names LSRULE_H2L3_SW \
    -location to \
    -cells PTLVLHLD2BWP \
    -prefix CPF_LS_SW


create_level_shifter_rule -name LSRULE_L2H2  \
    -from PDram \
    -to   PDdefault


create_level_shifter_rule -name LSRULE_H2L_PLL \
    -from PDpll


create_level_shifter_rule -name LSRULE_L2H3
    -from PDdefault   \
    -to   PDpll


###########################################################################
## create power net
###########################################################################
create_power_nets  -nets VDD -voltage {0.72:0.81:0.09}
create_power_nets  -nets VDD_sw -internal -voltage {0.72:0.81:0.09}
```

```
create_power_nets  -nets VDDL -voltage 0.72
create_power_nets  -nets VDDL_sw -internal -voltage 0.72


create_power_nets  -nets Avdd -voltage 0.81


create_power_nets -nets VDD_IO -voltage {0.72:0.81:0.09} \
    -external_shutoff_condition {io_shutoff_ack}


create_ground_nets -nets Avss -voltage 0
create_ground_nets -nets VSS -voltage 0


########################################################################
## create analysis view
########################################################################
create_operating_corner -name PMdvfs2_bc \
    -process 1 -temperature 0 -voltage 0.88  \
    -library_set bc_0v72
create_operating_corner -name PMdvfs1_bc \
    -process 1 -temperature 0 -voltage 0.99 \
    -library_set bc_0v81
create_operating_corner -name PMdvfs1_wc \
    -process 1 -temperature 125 -voltage 0.81 \
    -library_set wc_0v81
create_operating_corner -name PMdvfs2_wc \
    -process 1 -temperature 125 -voltage 0.72 \
    -library_set wc_0v72


create_analysis_view -name AV_dvfs1_BC -mode PMdvfs1    \
    -domain_corners {PDpll@PMdvfs1_bc PDdefault@PMdvfs1_bc PDtdsp@PMdvfs1_bc \
                     PDram@PMdvfs2_bc PDshutoff_io@PMdvfs1_bc}
create_analysis_view -name AV_dvfs1_WC -mode PMdvfs1 \
    -domain_corners {PDpll@PMdvfs1_wc PDdefault@PMdvfs1_wc PDtdsp@PMdvfs1_wc \
                     PDram@PMdvfs2_wc PDshutoff_io@PMdvfs1_wc}
create_analysis_view -name AV_dvfs1_off_BC -mode PMdvfs1_off \
    -domain_corners {PDpll@PMdvfs1_bc PDdefault@PMdvfs1_bc \
                     PDshutoff_io@PMdvfs1_bc}
create_analysis_view -name AV_dvfs1_off_WC -mode PMdvfs1_off \
    -domain_corners {PDpll@PMdvfs1_wc PDdefault@PMdvfs1_wc \
                     PDshutoff_io@PMdvfs1_wc}
create_analysis_view -name AV_dvfs1_shutoffio_off_BC -mode PMdvfs1_shutoffio_off \
    -domain_corners {PDpll@PMdvfs1_bc PDdefault@PMdvfs1_bc}
create_analysis_view -name AV_dvfs1_shutoffio_off_WC -mode PMdvfs1_shutoffio_off \
    -domain_corners {PDpll@PMdvfs1_wc PDdefault@PMdvfs1_wc}
create_analysis_view -name AV_dvfs2_BC -mode PMdvfs2 \
    -domain_corners {PDpll@PMdvfs1_bc PDdefault@PMdvfs2_bc PDtdsp@PMdvfs2_bc \
                     PDram@PMdvfs2_bc PDshutoff_io@PMdvfs2_bc}
create_analysis_view -name AV_dvfs2_WC -mode PMdvfs2 \
    -domain_corners {PDpll@PMdvfs1_wc PDdefault@PMdvfs2_wc PDtdsp@PMdvfs2_wc \
                     PDram@PMdvfs2_wc PDshutoff_io@PMdvfs2_wc}
```

```
create_analysis_view -name AV_PMdvfs2_off_BC -mode PMdvfs2_off \
    -domain_corners {PDpll@PMdvfs1_bc PDdefault@PMdvfs2_bc \
                     PDshutoff_io@PMdvfs2_bc}
create_analysis_view -name AV_PMdvfs2_off_WC -mode PMdvfs2_off \
    -domain_corners {PDpll@PMdvfs1_wc PDdefault@PMdvfs2_wc \
                     PDshutoff_io@PMdvfs2_wc}
create_analysis_view -name AV_dvfs2_shutoffio_off_BC -mode PMdvfs2_shutoffio_off \
    -domain_corners {PDpll@PMdvfs1_bc PDdefault@PMdvfs2_bc}
create_analysis_view -name AV_dvfs2_shutoffio_off_WC -mode PMdvfs2_shutoffio_off \
    -domain_corners {PDpll@PMdvfs1_wc PDdefault@PMdvfs2_wc}
create_analysis_view -name AV_scan_BC -mode PMscan \
    -domain_corners {PDpll@PMdvfs1_bc PDdefault@PMdvfs1_bc PDtdsp@PMdvfs1_bc \
                     PDram@PMdvfs2_bc PDshutoff_io@PMdvfs1_bc}
create_analysis_view -name AV_scan_WC -mode PMscan \
    -domain_corners {PDpll@PMdvfs1_wc PDdefault@PMdvfs1_wc PDtdsp@PMdvfs1_wc\
                     PDram@PMdvfs2_wc PDshutoff_io@PMdvfs1_wc}
########################################################################
##power switch, level shifter, isolation cell
########################################################################
create_power_switch_rule -name PDram_SW -domain PDram -external_power_net VDDL
update_power_switch_rule -name PDram_SW \
    -cells HDRDID1BWPHVT \
    -prefix CDN_SW_RAM
create_power_switch_rule -name PDtdsp_SW -domain PDtdsp -external_power_net VDD
update_power_switch_rule -name PDtdsp_SW \
    -cells HDRDID1BWPHVT \
    -prefix CDN_SW_TDSP
end_design
```

Macro CPF — ram.cpf

```
set_cpf_version 1.1
set_macro_model ram_256x16A
# This is a special RAM that retains its memory when it is powered down
# RAM has one power domain
create_power_domain -name RAM_DEFAULT -default \
    -boundary_ports {A* D* CLK CEN WEN Q*}
update_power_domain -name RAM_DEFAULT -primary_power_net VDD \
    -primary_ground_net VSS
# incomplete iso rule specified: the required iso value at inputs
# when driving domain is off
create_isolation_rule -name RAM_iso -isolation_output high -to RAM_DEFAULT
    -pins {A* D* CLK CEN WEN}
create_state_retention_rule -name RAM_ret -instances mem* -save_edge !CLK
end_macro_model
```

Soft IP CPF — tdsp.cpf

```
set_cpf_version 1.1
set_design tdsp_core -ports { retention_save retention_restore }

include tech.cpf
###########################################################################
## create power domains
###########################################################################
create_power_domain -name PDtdsp_block -default
update_power_domain -name PDtdsp_block -primary_power_net VDDtdsp_SW \
    -primary_ground_net VSStdsp
create_nominal_condition -name nom_0v72 -voltage 0.72
update_nominal_condition -name nom_0v72 -library_set wc_0v72
create_nominal_condition -name off -voltage 0
create_isolation_rule -name PDtdsp_iso -to PDtdsp_block -isolation_output low
###########################################################################
## create power modes
###########################################################################
create_power_mode -name PMtdsp_ON \
    -domain_conditions {PDtdsp_block@nom_0v72} \
    -default
create_power_mode -name PMtdsp_OFF \
    -domain_conditions {PDtdsp_block@off}
###########################################################################
## low power rules
###########################################################################
create_state_retention_rule -name PDtdsp_retention_rule \
    -domain PDtdsp_block  -save_edge retention_save \
    -restore_edge {!retention_restore}
update_state_retention_rules -names PDtdsp_retention_rule \
    -cell_type "ballon_latch"


end_design
```

Steps to Create the CPF File

Note: The example shown in Figure 4-1 on page 119 is used throughout this section.

Understanding the CPF file of the Macro Cell

A macro cell can have a behavioral model to describe its functionality. Implementation and verification tools therefore have to rely on the CPF modeling of the internal power network using the boundary ports. This section shows some of the constructs you can expect in a CPF macro model.

For the complete CPF macro model used in this design, refer to Macro CPF — ram.cpf.

1. The definition of a CPF macro model starts with a `set_macro_model` command that specifies the name of the library cell that represents the macro cell.

   For the macro model used in the design in Figure 4-1 on page 119:

   ```
   set_macro_model ram_256x16A
   ```

2. The definition ends with an `end_macro_model` command.

   The commands `set_macro_model` and `end_macro_model` delimit the scope of a macro model description.

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

3. The CPF macro model lists all the power domains that the macro cell contains.

   Only the (boundary) input and output pins of the macro cell are visible to the outside world. The input and output pins that belong to a specific power domain are specified through the `-boundary_ports` option of the corresponding `create_power_domain` command.

   The domain definition indicates whether the domain is always on or switchable. For switchable domains, the definition also indicates how the domain is powered down.

   The CPF macro model used in the design in Figure 4-1 on page 119 has one power domain. All of its boundary pins belong to this domain. The power domain is further unswitched.

   ```
   create_power_domain -name RAM_DEFAULT -default \
       -boundary_ports {A* D* CLK CEN WEN Q*}
   ```

   If the pin is an input pin, the power domain driven by the pin is the specified domain. If the pin is an output pin, the domain driving the pin is the specified power domain.

4. The CPF macro model also specifies for each of its power domains the boundary ports that provide the power supply to that domain.

   For the CPF macro model used in the design in Figure 4-1 on page 119:

   ```
   update_power_domain -name RAM_DEFAULT -primary_power_net VDD \
       -primary_ground_net VSS
   ```

5. If the macro cell has switchable power domains, its CPF macro model can contain isolation rules.

   A complete isolation rule in a macro cell describes the isolation logic implemented inside the macro cell. Complete rules contain the isolation condition and isolation output. Complete rules are optional but can be specified for documentation purposes.

   IP blocks can have special requirements for input ports. For example, when the domains 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, isolation rules without enable conditions (neither `-isolation_condition` nor `-no_condition` option specified) are specified. Such isolation rules are called *incomplete* isolation rules.

   For the CPF macro model used in the design in Figure 4-1 on page 119, an incomplete isolation rule specifies the required isolation value at its inputs when the driving domain is powered down.

   ```
   create_isolation_rule -name RAM_iso -to RAM_DEFAULT -isolation_output low \
       -pins {A* D* CLK CEN WEN}
   ```

   Note: For more information on how incomplete isolation rules are addressed when the macro is used, refer to “Resolve the precedence of the top-level and block-level rules” in Instantiating the Macro Cell in the Top-Level CPF and Loading the Corresponding CPF.

6. If the macro cell has switchable power domains, the CPF macro model can contain state retention rules.

   For the macro cell in Figure 4-1 on page 119, one state retention rule is defined.

   ```
   create_state_retention_rule -name RAM_ret -instances mem* -save_edge !CLK
   ```

   In this example, the memory array register names start with `mem`.

7. If the macro cell contains several switchable power domains, the CPF macro model can list its power modes. This allows the verification tools to check consistency between the modes defined at the top-level and in the macro to ensure that the macro cell is correctly integrated in the design.

Tip

Because a macro model is already implemented, there is no need to include a CPF file with technology-related information.

For more information on macro modeling, refer to Modeling a Macro Cell in the *Common Power Format Language Reference*.

Understanding the CPF file of the Soft IP

As opposed to a macro cell, the power intent of a soft IP still needs to be implemented. A soft IP can therefore be seen as an independent small design.

For the complete CPF of the soft block used in this design, refer to Soft IP CPF — tdsp.cpf.

1. The definition of a soft IP starts with a `set_design` command that specifies the name of the module that represents the soft IP.

   If the soft IP requires *virtual* ports (additional ports that do not exist in the definition of this module before implementation) for the control signals of the low power logic such as isolation logic, state-retention logic, and so on, these ports are specified with the `-ports` option of the `set_design` command.

   For the soft block used in the design in Figure 4-1 on page 119:

   ```
   set_design tdsp_core -ports { retention_save retention_restore }
   ```

   The `retention_save` and `retention_restore` ports are the virtual ports of the soft IP used in this design.

2. The definition ends with an `end_design` command.

3. All commands between `set_design` and `end_design` describe the internal power intent of the soft IP.

   The constructs used depend on the low power technique used. For more information, see the Chapter 2, “Creating a CPF File.”

Creating the CPF File for the Top-Level Design

The CPF of the top-level design contains the following information:

- Definition of the power domains at the top level
- Definition of the nominal conditions at the top level
- Definition of the power modes at the top level
- Association of the library sets with the nominal conditions
- **Instantiating the Macro Cell in the Top-Level CPF and Loading the Corresponding CPF**
- **Instantiating a Soft IP in the Top-Level CPF and Loading the Corresponding CPF**
- Specification of the rules at the top-level
- Specification of timing constraints, power targets, switching activities per mode
- Declaration of the power and ground nets
- Declaration of the global connections
- Addition of implementation information to the rules
- Specification of the operating corners
- Specification of the analysis views

The steps shown in **bold** are unique to the hierarchical flow and will be further explained here.

For the complete CPF of the top-level design in Figure 4-1 on page 119, refer to Top CPF File.

Instantiating the Macro Cell in the Top-Level CPF and Loading the Corresponding CPF

When you instantiate a macro cell in the design, you must indicate which CPF macro model applies to the specified instance. This can be done in one of the following ways:

- First load the CPF macro model, then use the `set_instance` command with the `-model` option to reference the CPF macro model.

  The CPF macro model can be included

  - Explicitly (see Example 4-1 on page 140)
  - Implicitly using the `include` command (see Example 4-2 on page 140)

- Use the `set_instance` command without the `-model` option, immediately followed by the CPF macro model.

  The CPF macro model can be included

  - Explicitly (see Example 4-3 on page 141)
  - Implicitly using the `include` command (see Example 4-4 on page 141)

Example 4-1 Explicit CPF Specification of Macro Loaded before Macro Instantiation

```
set_macro_model ram_256x16A
...
end_macro_model
set_design dtmf_chip
create_power_domain ...
create_nominal_condition ...
update_nominal_condition ...
create_power_mode...
set_instance DTMF_INST/RAM_128x16_TEST_INST1/RAM_128x16_INST -model ram_256x16A \
    -domain_mapping { {RAM_DEFAULT PDtdsp} }
...
end_design
```

Example 4-2 Implicit CPF Specification of Macro Loaded before Macro Instantiation

```
include ram.cpf
set_design dtmf_chip
create_power_domain ...
create_nominal_condition ...
update_nominal_condition ...
create_power_mode...
set_instance DTMF_INST/RAM_128x16_TEST_INST1/RAM_128x16_INST -model ram_256x16A \
    -domain_mapping { {RAM_DEFAULT PDtdsp} }
...
end_design
```

Example 4-3 Explicit CPF Specification of Macro Loaded after Macro Instantiation

```
set_design dtmf_chip
create_power_domain ...
create_nominal_condition ...
update_nominal_condition ...
create_power_mode...
set_instance DTMF_INST/RAM_128x16_TEST_INST1/RAM_128x16_INST \
    -domain_mapping { {RAM_DEFAULT PDtdsp} }
set_macro_model ram_256x16A
...
end_macro_model
...
end_design
```

Example 4-4 Implicit CPF Specification of Macro Loaded after Macro Instantiation

```
set_design dtmf_chip
create_power_domain ...
create_nominal_condition ...
update_nominal_condition ...
create_power_mode...
set_instance DTMF_INST/RAM_128x16_TEST_INST1/RAM_128x16_INST \
    -domain_mapping { {RAM_DEFAULT PDtdsp} }
include ram.cpf
...
end_design
```

When you instantiate a macro cell in the design, you must also indicate how the domains of the CPF macro model must be mapped to the domains of the parent level using the `-domain_mapping` option of the `set_instance` command. For the example in Figure 4-1 on page 119:

```
set_instance DTMF_INST/RAM_128x16_TEST_INST1/RAM_128x16_INST \
    -domain_mapping { {RAM_DEFAULT PDtdsp} }
```

The `ram_256x16A` model contains one power domain, `RAM_DEFAULT`. The `set_instance` command indicates that this domain must be mapped to the `PDtdsp` domain.

By mapping the domains, the tools will be able to

- Connect the primary power and ground pins of the block-level domain to the primary power and ground nets of the corresponding domain specified at the top level.

  The primary power and ground pins of power domain `RAM_DEFAULT` are `VDD` and `VSS`, while the primary nets of power domain `PDtdsp` are `VDDL_sw` and `VSS`. This implies that `VDD` will be connected to `VDDL_sw`, while the `VSS` pin will be connected to the `VSS` net.

- Merge the elements of each block-level domain into the corresponding top-level domain.

  In this design, the `RAM_128x16_TEST_INST1` instance becomes a leaf instance in the `PDtdsp` domain.

- Merge the power modes of the CPF macro model with the power modes of the top level.

  In this design, the `ram_256x16A` model does not have a power mode defined, it will have the same power modes as the top-level domain `PDtdsp`.

- Resolve the precedence of the power domain settings.

  Power domain settings are also referred to as domain attributes. For more information, refer to Handling Domain Attributes after Domain Mapping in the *Common Power Format Language Reference.* In this design there are no domain attributes to be resolved.

- Resolve the precedence of the top-level and block-level rules.

  The CPF macro model used in this design has a state retention rule.

  ```
  create_state_retention_rule -name RAM_ret -instances mem* -save_edge !CLK
  ```

  The top-level domain `PDtdsp` has no state retention rules. Even if the domain did have a rule, the block-level rule would still overwrite the top-level rule.

  The CPF macro model used in this design also has an isolation rule.

  ```
  create_isolation_rule -name RAM_iso -isolation_output low -to RAM_DEFAULT
      -pins {A* D* CLK CEN WEN}
  ```

  The isolation rule `RAM_iso` is an incomplete rule and applies to the net segments that drive logic in the `RAM_DEFAULT` domain. It also requires that the output value of the isolation gates be low when isolation is in effect. The rule specifies the isolation requirements at the selected ports of the macro and this information can be used by the top-level CPF to generate the correct isolation rule at this port.

  The tools will check the power domain to which the leaf level driver of a net selected by the rule belongs.

  - If the leaf-level driver belongs to the same power domain as the port that the rule applies to, or an unswitched power domain, no isolation of that net segment is needed.
  - If the leaf-level driver belongs to a different switchable power domain, the tools will check if that power domain 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.

  In this design the macro cell is driven by leaf-level drivers in the `PDdefault` domain, which is a non-switchable domain.

Instantiating a Soft IP in the Top-Level CPF and Loading the Corresponding CPF

When you instantiate a soft block in the design, you must indicate which CPF specification applies to the specified instance. This can be done in one of the following ways:

- First load the CPF specification, then use the `set_instance` command with the `-design` option.

  The CPF specification can be included

  - Explicitly (see Example 4-5 on page 143)
  - Implicitly using the `include` command (see Example 4-6 on page 144)

- Use the `set_instance` command without the `-design` option, followed by the CPF specification.

  The CPF specification can be included

  - Explicitly (see Example 4-7 on page 144)
  - Implicitly using the `include` command (see Example 4-8 on page 144)

Example 4-5 Explicit CPF Specification of Soft IP Loaded before Soft IP Instantiation

```
set_design tdsp_core -ports { retention_save retention_restore }
...
end_design
set_design dtmf_chip
create_power_domain ...
create_nominal_condition ...
update_nominal_condition ...
create_power_mode...
set_instance DTMF_INST/TDSP_CORE_INST0 -design tdsp_core \
    -domain_mapping { {PDtdsp_block PDtdsp} } \
    -port_mapping { {retention_save DTMF_INST/PM_INST/state_retention_save} \
                {retention_restore DTMF_INST/PM_INST/state_retention_restore} }
...
end_design
```

Example 4-6 Implicit CPF Specification of Soft IP Loaded before Soft IP Instantiation

```
include tdsp.cpf
set_design dtmf_chip
create_power_domain ...
create_nominal_condition ...
update_nominal_condition ...
create_power_mode...
set_instance DTMF_INST/TDSP_CORE_INST0 -design tdsp_core \
    -domain_mapping { {PDtdsp_block PDtdsp} } \
    -port_mapping { {retention_save DTMF_INST/PM_INST/state_retention_save} \
                {retention_restore DTMF_INST/PM_INST/state_retention_restore} }
...
end_design
```

Example 4-7 Explicit CPF Specification of Soft IP Loaded after Soft IP Instantiation

```
set_design dtmf_chip
create_power_domain ...
create_nominal_condition ...
update_nominal_condition ...
create_power_mode...
set_instance DTMF_INST/TDSP_CORE_INST0 -domain_mapping { {PDtdsp_block PDtdsp} } \
    -port_mapping { {retention_save DTMF_INST/PM_INST/state_retention_save} \
                {retention_restore DTMF_INST/PM_INST/state_retention_restore} }

set_design tdsp_core -ports { retention_save retention_restore }
...
end_design
...
end_design
```

Example 4-8 Implicit CPF Specification of Soft IP Loaded after Soft IP Instantiation

```
set_design dtmf_chip
create_power_domain ...
create_nominal_condition ...
update_nominal_condition ...
create_power_mode...
set_instance DTMF_INST/TDSP_CORE_INST0 -domain_mapping { {PDtdsp_block PDtdsp} } \
    -port_mapping { {retention_save DTMF_INST/PM_INST/state_retention_save} \
                {retention_restore DTMF_INST/PM_INST/state_retention_restore} }

include tdsp.cpf
...
end_design
```

When you instantiate a soft block in the design, you must also indicate

- How the domains of the soft block must be mapped to the domains of the parent level using the `-domain_mapping` option of the `set_instance` command.
- How the virtual ports must be mapped to the parent-level drivers using the `-port_mapping` option of the `set_instance` command. If there are no virtual ports declared in the corresponding `set_design` command, this option is not needed.
- The values that must be assigned to the parameters declared in the block-level CPF using the `-parameter_mapping` option of the `set_instance` command. If there are no parameters declared in the corresponding `set_design` command, this option is not needed.

  Note: For an example of parameter mapping, see the Examples for the `set_instance` command.

The example in Figure 4-1 on page 119 has two instances of the same soft block:

```
set_instance DTMF_INST/TDSP_CORE_INST0 \
    -domain_mapping { {PDtdsp_block PDtdsp} } \
    -port_mapping { {retention_save DTMF_INST/PM_INST/state_retention_save} \
                {retention_restore DTMF_INST/PM_INST/state_retention_restore} }
set_instance DTMF_INST/TDSP_CORE_INST1 \
    -domain_mapping { {PDtdsp_block PDtdsp} } \
    -port_mapping { {retention_save DTMF_INST/PM_INST/state_retention_save} \
                {retention_restore DTMF_INST/PM_INST/state_retention_restore} }
```

The `tdsp_core` model contains only one power domain, `PDtdsp_block`, which is mapped to the `PDtdsp` power domain for both instances. The two virtual ports for state retention control signals are connected to two output pins of the power control module (`PM_INST`).

The `PDtdsp_block` domain is an external switchable block-level domain, so it can be mapped into any type of domain at the top level. For other cases of domain mapping, refer to Handling Power Domain Mapping in the Common Power Format Language Reference.

By mapping the domains, the tools will be able to

- Connect the primary power and ground nets of the block-level domain to the primary power and ground nets of the corresponding domain specified at the top level.

  The primary power and ground nets of power domain `PDtdsp_block` are `VDDtdsp_SW` and `VSStdsp`, while the primary nets of power domain `PDtdsp` are `VDDL_sw` and `VSS`. This implies that `VDDtdsp_SW` will be connected to `VDDL_sw`, while `VSStdsp` will be connected to the `VSS` net.
- Merge the elements of each block-level domain into the corresponding top-level domain.

Because the soft block In this design only has one block domain, the `TDSP_CORE_INST0` and `TDSP_CORE_INST1` instances becomes instances of the `PDtdsp` domain.

- Merge the power modes of the soft IP with the power modes of the top level.

  If all block-level domains are mapped into top-level domains, the top-level mode definitions also become the power modes for the soft block after the domain mapping.

  In this design, the `tdsp_core` model has two power modes defined. Because it's one power domain maps to a top-level power domain, the soft block will have the same power modes as the top-level domain `PDtdsp`.

  For more information, refer to Handling Power Modes after Domain Mapping in the *Common Power Format Language Reference*.

- Resolve the precedence of the power domain settings.

  Power domain settings are also referred to as domain attributes. For more information, refer to Handling Domain Attributes after Domain Mapping in the *Common Power Format Language Reference.* In this design there are no domain attributes to be resolved.

- Resolve the precedence of the top-level and block-level rules.

  The CPF specification of the soft IP used in this design has a state retention rule.

  ```
  create_state_retention_rule -name PDtdsp_retention_rule \
      -domain PDtdsp_block  -save_edge retention_save \
      -restore_edge {!retention_restore}
  ```

  The top-level domain `PDtdsp` has no state retention rules. Even if the domain did have a rule, the block-level rule would still overwrite the top-level rule.

  The CPF specification of the soft IP used in this design also has an isolation rule.

  ```
  create_isolation_rule -name PDtdsp_iso -to PDtdsp_block -isolation_output low
  ```

  The isolation rule `PDtdsp_iso` is an incomplete rule and applies to the net segments that drive logic in the `PDtdsp_block` domain. It also requires that the output value of the isolation gates be low when isolation is in effect. However to complete the rule, the isolation condition must be derived from the corresponding top-level domain.

  The tools will check the power domain to which the leaf level driver of a net selected by the rule belongs.

  - If the leaf-level driver belongs to the same power domain, no isolation of that net segment is needed.
  - If the leaf-level driver belongs to a different switchable power domain, the tools will check if that power domain 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.

In this design the IP instances, `TDSP_CORE_INST0` and `TDSP_CORE_INST1`, are driven by leaf-level drivers in the `PDdefault` domain, which is a non-switchable domain.

For more information refer to Precedence of Rules in the Hierarchical Flow in the *Common Power Format Language Reference*

5

Modeling Special Cells

Modeling Level Shifters

Types of Level Shifters

To pass signals between portions of the design that operate on different power or ground voltages, level shifters are needed.

CPF can describe many different types of level shifters. The following is a list of the most typical level shifters:

- power level shifters—pass signals between portions of the design that operate on different power voltages but the same ground voltages
- ground level shifters—pass signals between portions of the design that operate on different ground voltages but the same power voltages
- power and ground level shifters—pass signals between portions of the design that operate on different power and ground voltages
- enabled level shifters—level shifters with an enable pin which allows them to be used as isolation cell as well
- bypass level shifters—a level shifter whose level shifting functionality can be bypassed under certain conditions

In addition, CPF can model multi-stage and multi-bit level shifters.

All types of level shifters are defined using the `define_level_shifter_cell` command. The following sections indicate which command options to use for each type.

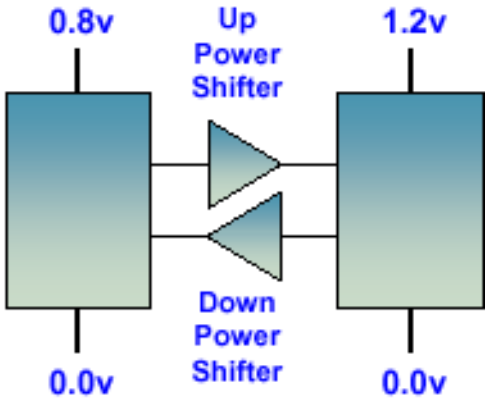
Modeling a Power Level Shifter

To model a power level shifter use the `define_level_shifter_cell` command with the following options.

```
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}
     [-direction {up|down|bidir}]
     [-input_power_pin LEF_power_pin] [-output_power_pin LEF_power_pin]
     |-ground LEF_ground_pin}
     [-valid_location {to | from | either}]
```

Figure5-1 shows a power domain at 0.8V and one at 1.2V. The ground voltage for both domains is 0.0V. In this case, for data signals going from the domain at 0.8V to the domain at 1.2V a power level shifter with direction **up** is needed, while for data signals going from the domain at 1.2V to the domain at 0.8V a power level shifter with direction **down** is needed.

Figure 5-1 Power Level Shifter



Sample commands to model these power level shifters are:

```
define_level_shifter_cell  -cells up_shift \
    -input_voltage_range 0.8:1.0 -output_voltage_range 1.0:1.2 \
    -input_power_pin VDD_IN -output_power_pin VDD_OUT -ground VSS_IN \
    -direction up  -valid_location from
define_level_shifter_cell  -cells down_shift \
    -input_voltage_range 1.0:1.2 -output_voltage_range 0.8:1.0 \
    -input_power_pin VDD_IN -output_power_pin VDD_OUT -ground VSS_IN \
    -direction down -valid_location from
```

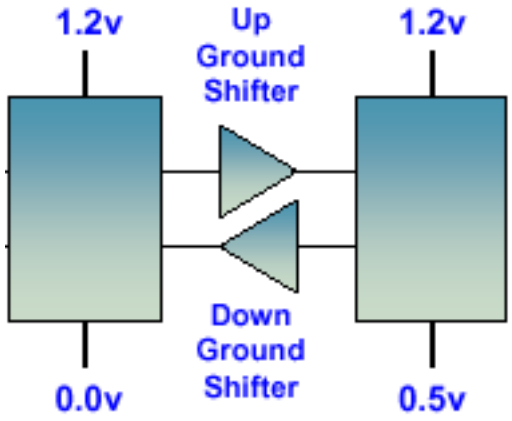
Modeling a Ground Level Shifter

To model ground level shifter use the `define_level_shifter_cell` command with the following options.

```
define_level_shifter_cell
     -cells cell_list [-library_set library_set]
     [-always_on_pins pin_list]
     -ground_input_voltage_range {voltage_list | voltage_range_list}
     -ground_output_voltage_range {voltage_list | voltage_range_list}
     [-direction {up|down|bidir}]
     [-input_ground_pin LEF_power_pin] [-output_ground_pin LEF_power_pin]
     [-power LEF_ground_pin]
     [-valid_location {to | from | either}]
```

The two power domains in Figure5-2 have the same power supply 1.2V. However, the ground voltage for the first domain is at 0.0V, while the ground voltage for the second domain is at 0.5V. For data signals going from the domain with ground voltage 0.0V to the domain with ground voltage 0.5V a ground level shifter with direction **up** is required. For data signals going from the domain with ground voltage 0.5V to the domain with ground voltage 0.0V a ground level shifter with direction **down** is required.

Figure 5-2 Ground Level Shifter



Sample commands to model these ground level shifters are:

```
define_level_shifter_cell  -cells up_shift \
    -ground_input_voltage_range 0.0:0.1 -ground_output_voltage_range 0.4:0.5 \
    -input_ground_pin VSS_IN -output_ground_pin VSS_OUT -power VDD_IN \
    -direction up  -valid_location from
define_level_shifter_cell  -cells down_shift \
    -ground_input_voltage_range 0.4:0.5 -ground_output_voltage_range 0.0:0.1 \
    -input_ground_pin VSS_IN -output_ground_pin VDD_OUT -power VDD_IN \
    -direction down -valid_location from
```

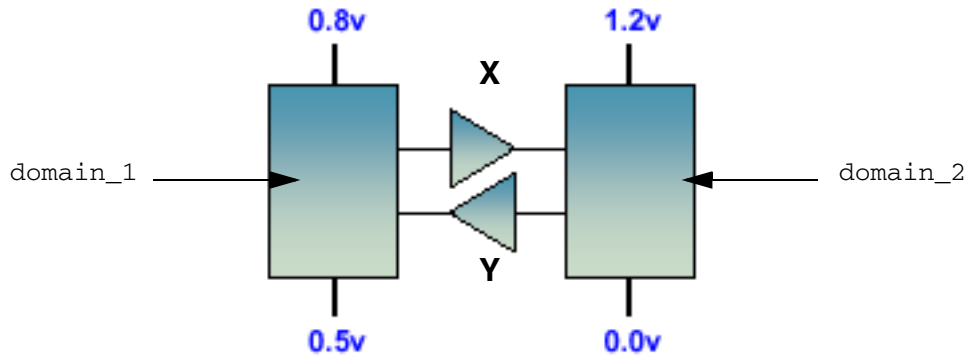
Modeling a Power and Ground Level Shifter

To model a power and ground level shifter, use two `define_level_shifter_cell` commands: one which defines the power level shifting and one which defines the ground level shifting. Both commands must point to the same cell(s), and must have the same value for the `-valid_location` option.

```
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}
     [-direction {up|down|bidir}]
     [-input_power_pin LEF_power_pin] [-output_power_pin LEF_power_pin]
     [-ground LEF_ground_pin]
     [-valid_location {to | from | either}]
define_level_shifter_cell
     -cells cell_list [-library_set library_set]
     [-always_on_pins pin_list]
     -ground_input_voltage_range {voltage_list | voltage_range_list}
     -ground_output_voltage_range {voltage_list | voltage_range_list}
     [-direction {up|down|bidir}]
     [-input_ground_pin LEF_power_pin] [-output_ground_pin LEF_power_pin]
     [-power LEF_power_pin]
     [-valid_location {to | from | either}]
```

The two power domains in Figure 5-3 have different power and ground voltages. Going from `domain_1` to the `domain_2` requires an power level shifter in the up direction and a ground level shifter in the down direction. Going from `domain_2` to `domain_1` requires a power level shifter in the down direction and a ground level shifter in the up direction.

Figure 5-3 Power and Ground Level Shifter



The following commands models the power and ground level shifter to go from `domain_1` to `domain_2`:

```
define_level_shifter_cell  -cells X \
    -input_voltage_range 0.8:1.0 -output_voltage_range 1.0:1.2 \
    -input_power_pin VDD_IN -output_power_pin VDD_OUT -ground VSS_IN \
    -direction up  -valid_location from

define_level_shifter_cell  -cells X
    -ground_input_voltage_range 0.4:0.5 -ground_output_voltage_range 0.0:0.1 \
    -input_ground_pin VSS_IN -output_ground_pin VSS_OUT -power VDD_OUT \
    -direction down -valid_location from
```

Both commands refer to the same cell, use the same valid location and have opposite directions for the power and ground level shifting.

The following commands models the power and ground level shift to go from `domain_2` to `domain_1`:

```
define_level_shifter_cell  -cells Y \
    -input_voltage_range 1.0:1.2 -output_voltage_range 0.8:1.0 \
    -input_power_pin VDD_OUT -output_power_pin VDD_IN -ground VSS_OUT \
    -direction down -valid_location to

define_level_shifter_cell  -cells Y \
    -ground_input_voltage_range 0.0:0.1 -ground_output_voltage_range 0.4:0.5 \
    -input_ground_pin VSS_OUT -output_ground_pin VSS_OUT -power VDD_OUT \
    -direction up -valid_location to
```

Modeling an Enabled Level Shifter

To model a level shifter that can also be used for isolation purposes, use the `define_level_shifter_cell` command with the `-enable` option.

This type of cell uses an enable pin to control the voltage shifting. Typically the enable pin is related to the output supplies of the level shifter. In other words, the enable control needs to have the same voltage as the to domain. If both domains are powered on, you can tie the enable to be always active and it works as a level shifter.

Tip

To model an isolation-level-shifter combo cell, see Modeling an Isolation-Level Shifter Combo Cell on page 174.

Modeling an Enabled Power Level Shifter

Assume that the power level shifter shown in Figure 5-1 on page 151 also has an enable pin to enable the level-shifting functionality. If the enable signal is inactive for the level shifting purposes, it protects the level shifter cell when the input power supply is powered down and causes the output to be a specific logic value determined by its functionality. However, the driver of the level shifter data pin is not protected. For such a cell to be used for isolation purposes when the driving domain is switched off using a header power switch, its input power pin must be connected to the primary power net of the driving domain. In this case the definition should be adjusted as follows:

```
define_level_shifter_cell  -cells up_shift \
    -input_voltage_range 0.8:1.0 -output_voltage_range 1.0:1.2 \
    -input_power_pin VDD_IN -output_power_pin VDD_OUT -ground VSS_IN \
    -direction up  -valid_location from \
    -enable en
```

Where `en` is the enable pin that controls the power shifting. Typically the enable pin is related to the output supplies of the level shifter.

Modeling an Enabled Ground Level Shifter

Assume that the ground level shifter shown in Figure 5-2 on page 152 also has an enable pin to enable the level-shifting functionality. If the enable signal is inactive for the level shifting purposes, it protects the level shifter cell when the input ground supply is shut off and causes the output to be a specific logic value determined by its functionality. However, the driver of the level shifter data pin is not protected. For such a cell to be used for isolation purposes when the driving domain is switched off using a footer ground switch, its input ground pin must be connected to the primary ground net of the driving domain. In this case the definition should be adjusted as follows:

```
define_level_shifter_cell  -cells down_shift \
    -ground_input_voltage_range 0.4:0.5 -ground_output_voltage_range 0.0:0.1 \
    -input_ground_pin VSS_IN -output_ground_pin VDD_OUT -power VDD_IN \
    -direction down -valid_location from \
    -enable en
```

Where `en` is the enable pin that controls the ground shifting. Typically the enable pin is related to the output supplies of the level shifter.

Modeling an Enabled Power and Ground Level Shifter

For power and ground level shifter cells, the enable pin needs to be specified with the definition of the power (ground) level shifter depending on whether the pin needs to control the power (ground) shifting. The layout of the cell determines whether it can be used in a domain that is power-switched (using a header switch cell) or ground-switched (using a footer switch cell). This section will illustrate a couple of examples.

In all examples, data signal `A` is driven by a power domain whose power and ground supply connects to `VDD_IN` and `VSS_IN`, respectively. The data output signal `Y` feeds in to a domain whose power and ground supply connects to `VDD_OUT` and `VSS_OUT`, respectively. This requires a power and ground level shifter cell. In the following examples, the following relations apply:

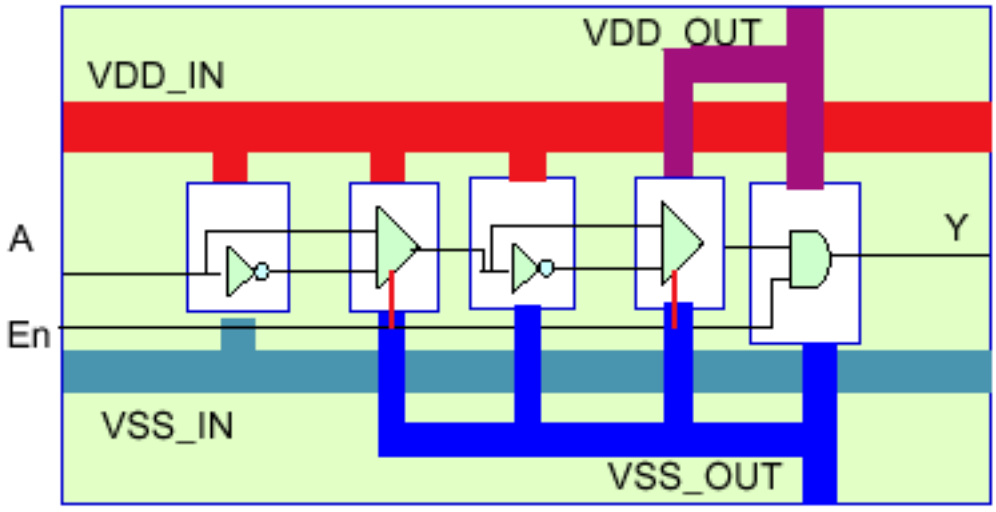
```
VSS_OUT < VSS_IN
VDD_OUT > VDD_IN
```

In addition, because the power domain driving `A` can be switched off, the level shifter must also be an enabled shifter. The required type of the enabled power and ground shifter will depend on whether the power domain is *power* or *ground* switched and on where the enabling logic gets is power and ground supply from.

Example 1

The type of level shifter shown in Figure 5-4, can be used when the power domain driving signal A is ground switched. The power and ground of the enabling logic are connected to `VDD_OUT` and `VSS_OUT`, respectively.

Figure 5-4 Type 1: Ground Shifter in Input Stage and Enabling Logic in Output Stage



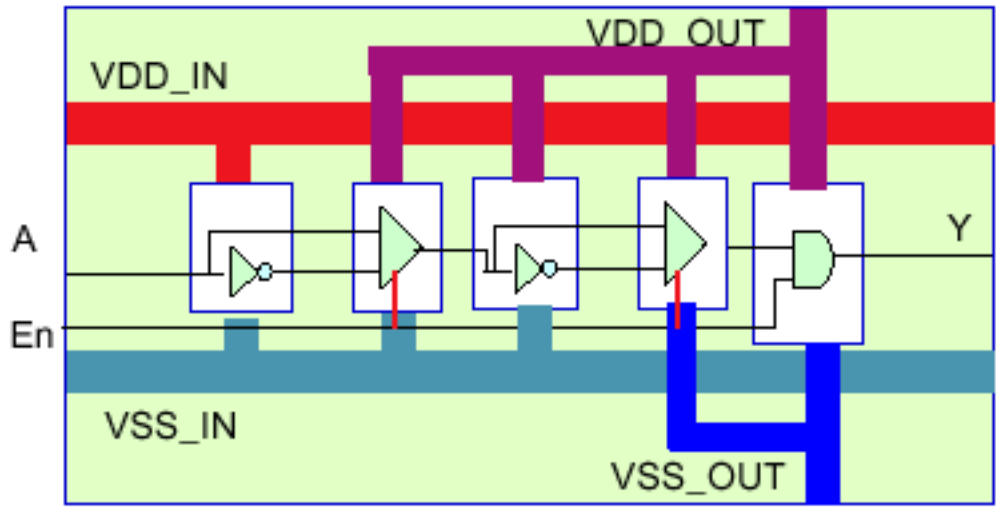
The following CPF commands describe this type 1 power and ground enabled level shifter:

```
define_level_shifter_cell  -cells X \
    -ground_input_voltage_range 0.2:0.3 -ground_output_voltage_range 0.1:0.2 \
    -input_ground_pin VSS_IN -output_ground_pin VSS_OUT -power VDD_IN \
    -direction down -valid_location from

define_level_shifter_cell  -cells X \
    -input_voltage_range 0.8:1.0 -output_voltage_range 1.0:1.2 \
    -input_power_pin VDD_IN -output_power_pin VDD_OUT -ground VSS_OUT \
    -direction up  -valid_location from
    -enable en
```

Example 2

The type of level shifter shown in Figure 5-5, can be used when the power domain of signal A is power switched. The power and ground of the enabling logic are connected to `VDD_OUT` and `VSS_OUT`, respectively.

Figure 5-5 Type 2: Power Shifter in Input Stage and Enable Logic in Output Stage



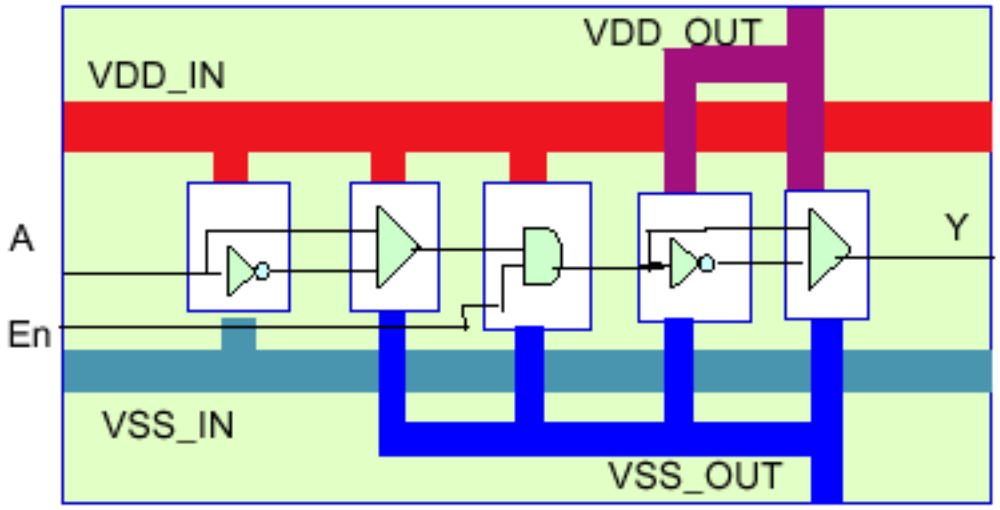
The following CPF commands describe this type 2 of power and ground enabled level shifter:

```
define_level_shifter_cell  -cells X \
    -ground_input_voltage_range 0.2:0.3 -ground_output_voltage_range 0.1:0.2 \
    -input_ground_pin VSS_IN -output_ground_pin VSS_OUT -power VDD_OUT \
    -direction down -valid_location from -enable en

define_level_shifter_cell  -cells X \
    -input_voltage_range 0.8:1.0 -output_voltage_range 1.0:1.2 \
    -input_power_pin VDD_IN -output_power_pin VDD_OUT -ground VSS_IN \
    -direction up  -valid_location from
```

Example 3

The type of level shifter shown in Figure 5-6, can be used when the power domain is ground switched. The power and ground of the enabling logic are connected to `VDD_IN` and `VSS_OUT`, respectively.

Figure 5-6 Type 3: Ground Shifter and Enable Logic in Input Stage



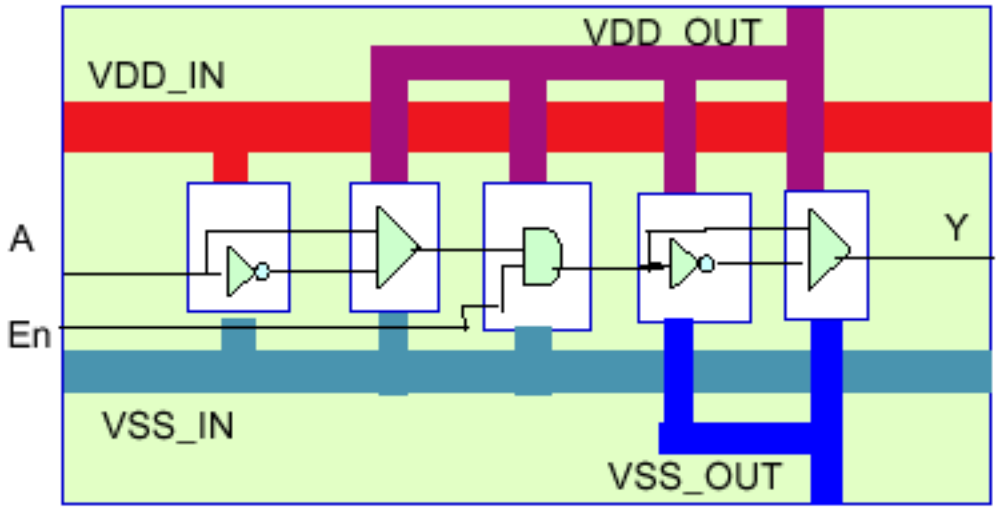
The following CPF commands describe this type 3 of power and ground enabled level shifter:

```
define_level_shifter_cell  -cells X \
    -ground_input_voltage_range 0.2:0.3 -ground_output_voltage_range 0.1:0.2 \
    -input_ground_pin VSS_IN -output_ground_pin VSS_OUT -power VDD_IN \
    -direction down -valid_location from -enable en

define_level_shifter_cell  -cells X \
    -input_voltage_range 0.8:1.0 -output_voltage_range 1.0:1.2 \
    -input_power_pin VDD_IN -output_power_pin VDD_OUT -ground VSS_OUT \
    -direction up  -valid_location from
```

Example 4

The type of level shifter shown in Figure 5-7, can be used when the power domain is power switched. The power and ground of the enabling logic are connected to `VDD_OUT` and `VSS_IN`, respectively.

Figure 5-7 Type 4: Power Shifter and Enable Logic in Input Stage



The following CPF commands describe this type 4 of power and ground enabled level shifter:

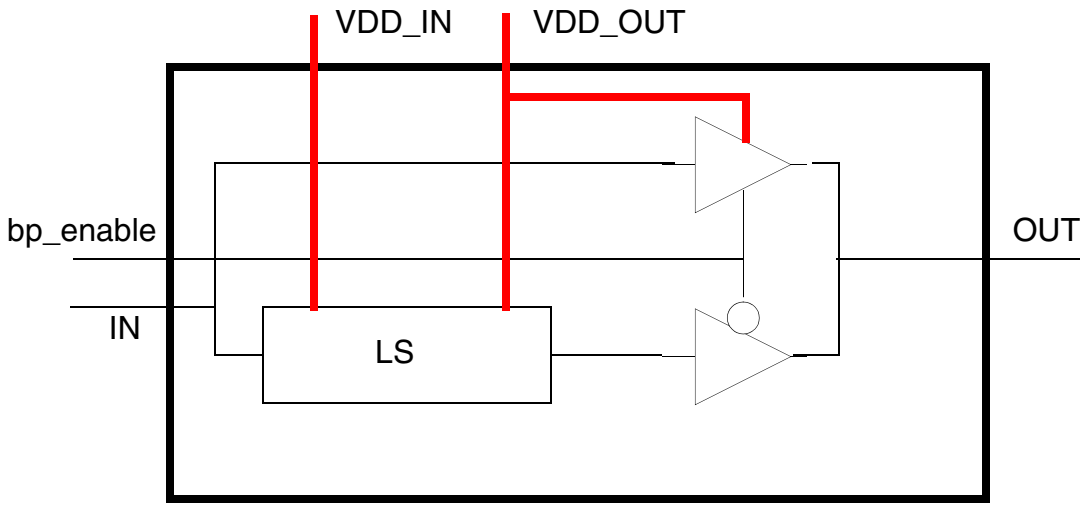
```
define_level_shifter_cell  -cells X \
    -ground_input_voltage_range 0.2:0.3 -ground_output_voltage_range 0.1:0.2 \
    -input_ground_pin VSS_IN -output_ground_pin VSS_OUT -power VDD_OUT \
    -direction down -valid_location from

define_level_shifter_cell  -cells X \
    -input_voltage_range 0.8:1.0 -output_voltage_range 1.0:1.2 \
    -input_power_pin VDD_IN -output_power_pin VDD_OUT -ground VSS_IN \
    -direction up  -valid_location from -enable en
```

Modeling a Bypass Level Shifter

To model a level shifter whose level shifting functionality can be bypassed under certain conditions, use the `define_level_shifter_cell` command with the `-bypass_enable` option.

An example of such a cell is shown in Figure5-8. When the `bp_enable` signal is `true`, the level shifting functionality is bypassed and signal OUT will be coming from the top buffer.

Figure 5-8 Bypass Level Shifter Cell



The following CPF command can be used to describe a bypass level shifter:

```
define_level_shifter_cell  -cells up_shift \
    -input_voltage_range 0.8:1.0 -output_voltage_range 1.0:1.2 \
    -input_power_pin VDD_IN -output_power_pin VDD_OUT -ground VSS \
    -direction up -valid_location from -bypass_enable bp_enable
```

To apply such a cell for a specific level shifter rule, you need to use the `-bypass_condition` option of the `create_level_shifter_rule` command to indicate the signal for the bypass enable pin for the cell.

Modeling a Multi-Stage Level Shifter

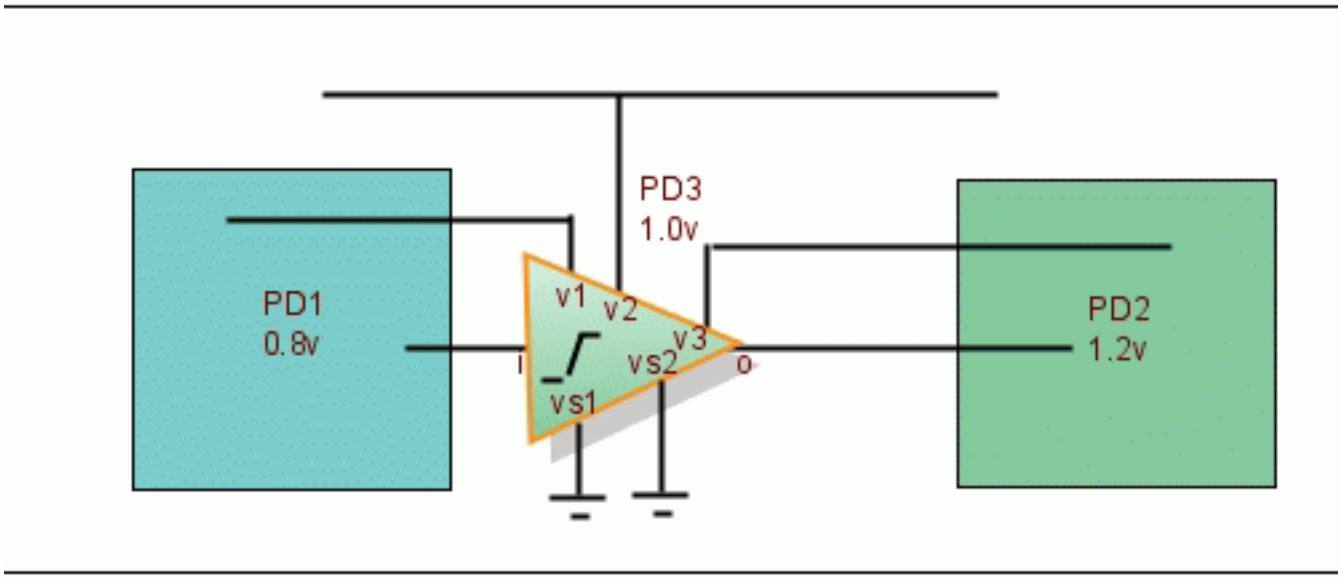
When the voltage difference between the driving (or originating) and receiving (or destination) power domains is large, multiple level shifters or a single multi-stage level shifter might be required.

To model a single multi-stage level shifter cell, define the level-shifter cell with the `-multi_stage` option in the `define_level_shifter_cell` command to identify the stage of the 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 to indicate the corresponding stage of this definition. A definition must not have the same stage defined twice.

An example of a single multi-stage level-shifter cell is shown in Figure 5-9.

Figure 5-9 Multi-Stage Level Shifter



The following CPF commands can be used to describe the single level shifter cell shown in Figure 5-9.

```
define_level_shifter_cell -cells fooA -multi_stage 1 -input_power_pin V1 \
-output_power_pin V2 -input_ground_pin VS1 -output_ground_pin VS2
define_level_shifter_cell -cells fooA -multi_stage 2 -input_power_pin V2 \
-input_ground_pin VS2 -output_voltage_pin V3 -output_ground_pin VS2
```

When you define the level shifter rule, you can indicate in the `update_level_shifter_rules` command whether to use a single multi-stage level-shifter cell or an ordered list of level-shifter cells:

- To use a single multi-stage level-shifter cell you must use the `-through` option to specify the subsequent domains for the multi-stage level shifting between the first domain (specified with `-from` in the `create_level_shifter_rule` command) and the last domain (specified with `-to` in the `create_level_shifter_rule` command).

  In this case, the cell must have been defined with the `-multi_stage` option in the `define_level_shifter_cell` command.

- To use multiple cells, you must specify an ordered list of N cell lists for the `-cells` option, where N is the number of stages in the level-shifting. Each list contains the cells appropriate for its stage.

Modeling a Multi-bit Level Shifter Cell

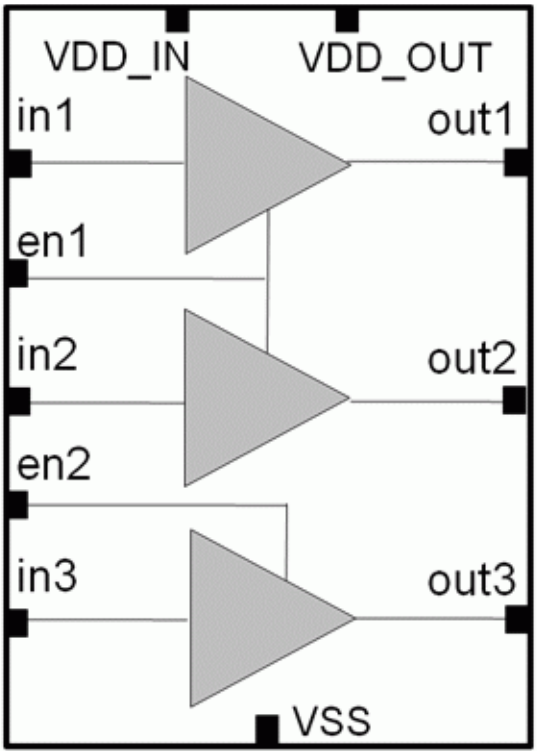
A multi-bit level shifter cell has multiple pairs of input and output pins with each pair serving as a single bit level shifter. An example is shown in Figure 5-10.

For the following multi-bit level shifter cells, there is no difference in modeling such a multi-bit cell with respect to the single bit level shifter cell:

- a multi-bit simple level shifter without an enable pin
- a multi-bit enable level shifter with the same enable pin for all bits

If the cell has different enable pins for the input and output pairs, you must model the cell using the `-pin_groups` option of the `define_level_shifter_cell` command.

Figure 5-10 Multi-Bit Level Shifter



The following CPF command can be used to describe the multi-bit level shifter cell shown in Figure 5-10.

```
define_level_shifter_cell -cells multi_bit_en \
    -input_voltage_range 0.8:1.0 -output_voltage_range 1.0:1.2 \
    -input_power_pin VDD_IN -output_power_pin VDD_OUT -ground VSS \
    -direction up -valid_location from \
    -pin_groups {{in1 out1 en1} {in2 out2 en1} {in3 out3 en2}}
```

Note: `-pin_groups` is a new option introduced in the SI2 CPF 2.0 version.

Modeling Isolation Cells

Types of Isolation Cells

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. CPF can describe many different types of isolation cells. Below is a list of the most typical isolation cells:

- Isolation cell to be placed in the unswitched domain
- Isolation cell to be used in a ground switchable domain
- Isolation cell to be used in a power switchable domain
- Isolation cells to be used in a a power or ground switchable domain
- Isolation cells without followpins that can be placed in any domain
- Isolation cells without an enable pin
- Isolation clamp cell
- Isolation-level shifter combo cell

All types of isolation cells are defined using the `define_isolation_cell` command. The following sections indicate which command options to use for each type.

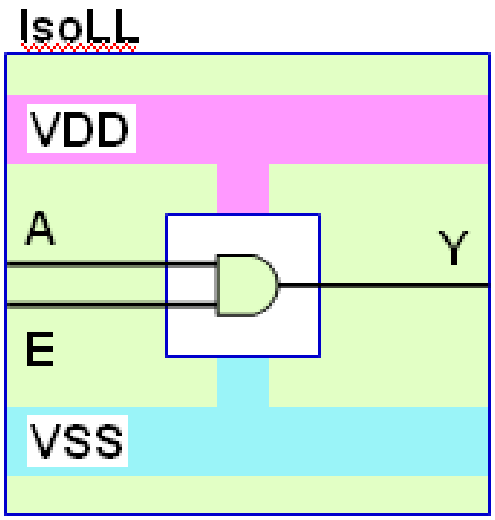
Modeling an Isolation Cell to be Placed in the Unswitched Domain

To model an isolation cell to be placed in an unswitched domain, use the `define_isolation_cell` command with the following options.

```
define_isolation_cell
     -cells cell_list [-library_set library_set]
     -power LEF_power_pin -ground LEF_ground_pin
     -valid_location on
     { -enable pin | -no_enable {high|low|hold} } [-non_dedicated]
```

Figure 5-11 shows an AND cell that can be used for isolation purposes.

Figure 5-11 Dedicated Isolation Cell in Unswitched Domain



The following command models the isolation cell in Figure 5-11:

```
define_isolation_cell \
    -cells IsoLL \
    -power VDD -ground VSS \
    -enable E \
    -valid_location on
```

Note: If you also want to use the cell in regular logic, you must add the `-non_dedicated` option. Non-dedicated cells can typically only be placed in the unswitched domain (`-valid_location on`).

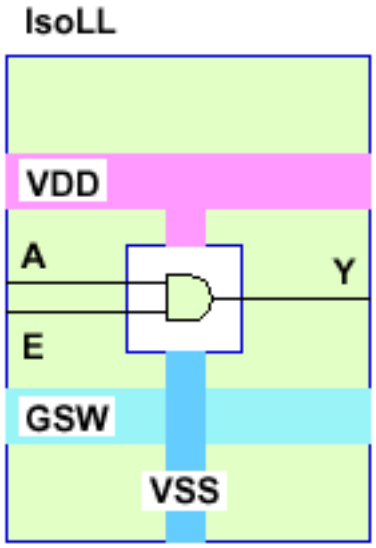
Modeling An Isolation Cell for Ground Switchable Domain

To model an isolation cell to be used in a ground switchable domain, use the `define_isolation_cell` command with the following options.

```
define_isolation_cell
     -cells cell_list [-library_set library_set]
     [-always_on_pins pin_list]
     -ground_switchable LEF_ground_pin
     -power LEF_power_pin -ground LEF_ground_pin
     [-valid_location { from | to | on | off}]
     { -enable pin | -no_enable {high|low|hold} } [-non_dedicated]
```

Figure 5-12 shows an AND cell that has the path from power to ground cut off on the ground side. This AND cell can only be used for isolation.

Figure 5-12 Dedicated Ground Switchable Isolation Cell



The following command models the isolation cell in Figure 5-12:

```
define_isolation_cell \
    -cells IsoLL \
    -ground_switchable GSW \
    -power VDD -ground VSS \
    -enable E \
    -valid_location from
```

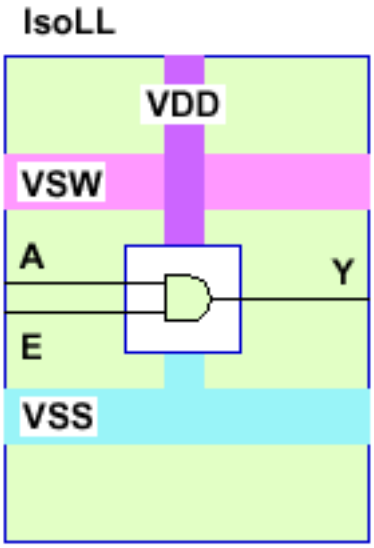
Modeling An Isolation Cell for Power Switchable Domain

To model an isolation cell to be used in a power switchable domain, use the `define_isolation_cell` command with the following options.

```
define_isolation_cell
     -cells cell_list [-library_set library_set]
     [-always_on_pins pin_list]
     -power_switchable LEF_power_pin
     -power LEF_power_pin -ground LEF_ground_pin
     [-valid_location { from | to | on | off}]
     { -enable pin | -no_enable {high|low|hold} }
```

Figure 5-13 shows an AND cell that has the path from power to ground cut off on the power side. This AND cell can only be used for isolation.

Figure 5-13 Dedicated Power Switchable Isolation Cell



The following command models the isolation cell in Figure 5-13:

```
define_isolation_cell \
    -cells IsoLL \
    -power_switchable VSW \
    -power VDD -ground VSS \
    -enable E \
    -valid_location from
```

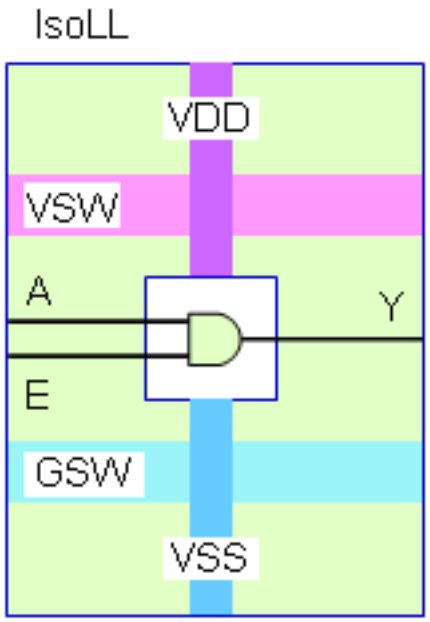
Modeling An Isolation Cell for Power and Ground Switchable Domains

To model an isolation cell to be used in a power and ground switchable domain, use the `define_isolation_cell` command with the following options.

```
define_isolation_cell
     -cells cell_list [-library_set library_set]
     [-always_on_pins 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}]
     { -enable pin | -no_enable {high|low|hold} }
```

Figure 5-14 shows an AND cell that has the path from power to ground cut off on the power and ground sides. This AND cell can only be used for isolation.

Figure 5-14 Dedicated Power and Ground Switchable Isolation Cell



The following command models the isolation cell in Figure 5-14:

```
define_isolation_cell \
    -cells IsoLL \
    -power_switchable VSW -ground_switchable GSW \
    -power VDD -ground VSS \
    -enable E \
    -valid_location from
```

Modeling An Isolation Cells that Can Be Placed in Any Domain

To model an isolation cell to be used in any domain, use the `define_isolation_cell` command with the following options.

```
define_isolation_cell
     -cells cell_list [-library_set library_set]
     [-always_on_pins pin_list]
     -power LEF_power_pin -ground LEF_ground_pin
     -valid_location any
     { -enable pin | -no_enable {high|low|hold} }
     [-non_dedicated]
```

Modeling An Isolation Cell Without Enable Pin

There are special isolation cells which do not have an enable pin but still can clamp output to a logic value when primary power supply is switched off. To model such a cell, use `define_isolation_cell` command with the following options.

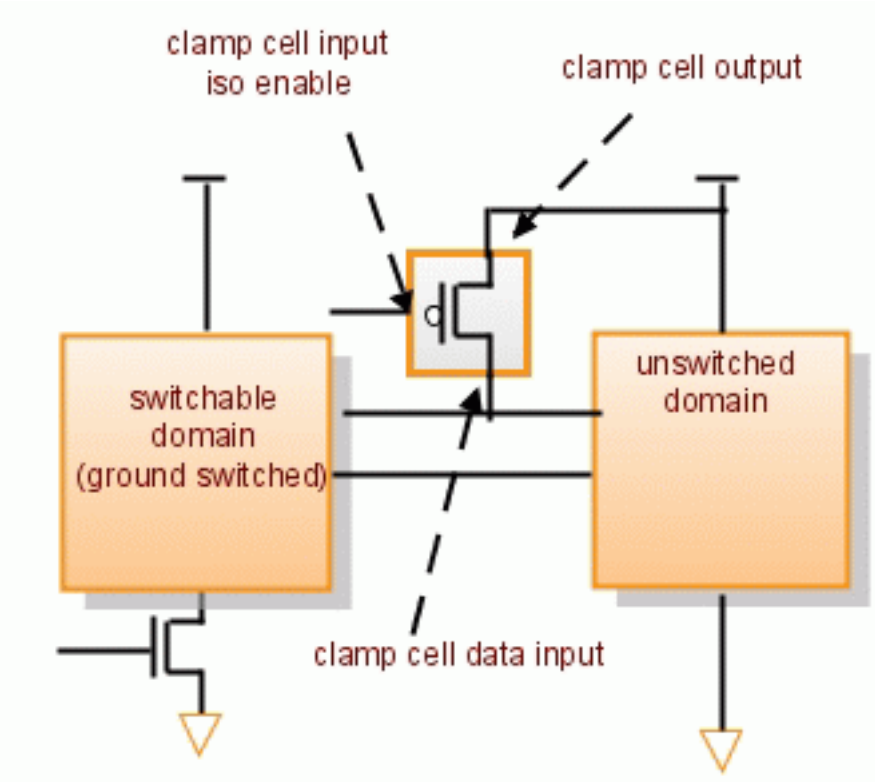
```
define_isolation_cell
     -cells cell_list [-library_set library_set]
     [-always_on_pins pin_list]
     [-power_switchable LEF_power_pin] [-ground_switchable LEF_ground_pin]
     [-power LEF_power_pin] [-ground LEF_ground_pin]
     [-valid_location { from | to | on | off | any}]
     -no_enable {high|low|hold}
```

Note: You can only infer such cells in a design by specifying the `create_isolation_rule` command with `-no_condition` option.

Modeling An Isolation Clamp Cell

An isolation clamp high cell is a simple PMOS transistor with the gate input being used as the enable pin. When its driver is switched off by a ground switch and the enable pin has value 0, the connected net can be clamped to a logic high value as shown in Figure 5-15.

Figure 5-15 Isolation clamp high cell

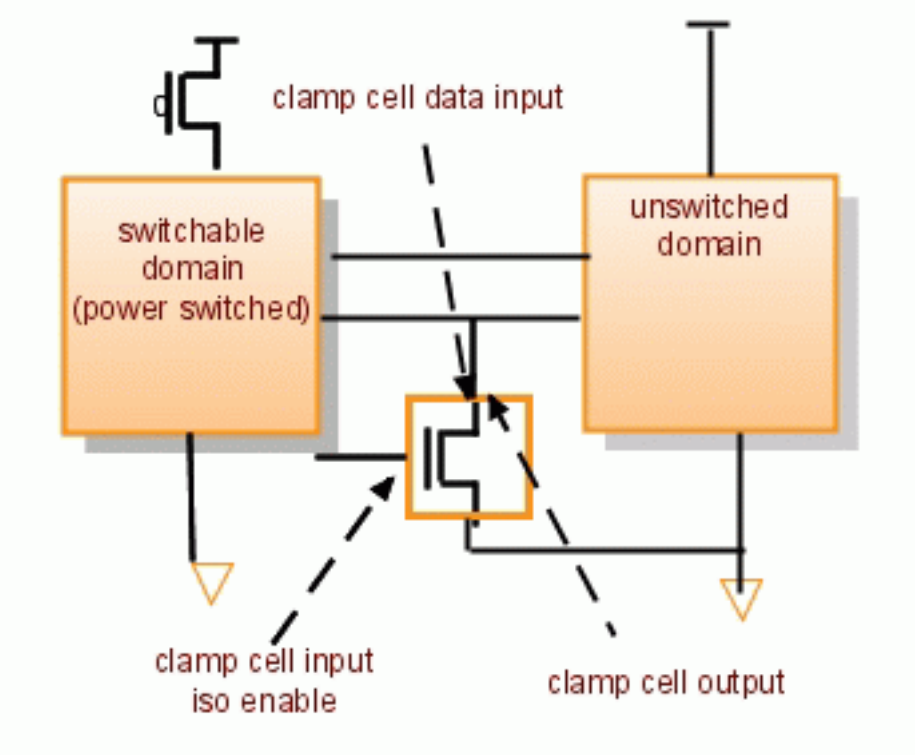


To model an isolation clamp high cell, use the `define_isolation_cell` command with the following options.

```
define_isolation_cell
     -cells cell_list [-library_set library_set]
     [-always_on_pins pin_list]
     -power LEF_power_pin
     [-valid_location { from | to | on | off | any}]
     -enable pin -clamp high
     [-non_dedicated]
```

An isolation clamp low cell is a simple NMOS transistor with the gate input being used as the enable pin. When its driver is switched off by a power switch and the enable pin has value 1, the connected net can be clamped to a logic low value as shown in Figure 5-16 on page 173.

Figure 5-16 Isolation clamp low cell



To model an isolation clamp low cell, use the `define_isolation_cell` command with the following options.

```
define_isolation_cell
     -cells cell_list [-library_set library_set]
     [-always_on_pins pin_list]
     -ground LEF_ground_pin
     [-valid_location { from | to | on | off | any}]
     -enable pin -clamp low
     [-non_dedicated]
```

Due to its special connectivity requirement, to apply such a power or ground clamp cell for a specific isolation rule, you need to use `-isolation_output` option with either the `clamp_high` or `clamp_low value` in the `create_isolation_rule` command.

Modeling an Isolation-Level Shifter Combo Cell

A combo cell isolates or protects the input when the driving logic is powered down and generates an output isolation value at the same voltage as the output supply of the cell. Typically the enable pin is related to the input supplies of the cell.

The most common combo cells are the isolation cells with high to low shifting capabilities.

To model a combo cell you need two commands. For example to model an isolation cell for power switchable domain that is also a power level shifter, use the following definitions:

```
define_isolation_cell
     -cells cell_list [-library_set library_set]
     [-always_on_pins pin_list]
     -power_switchable LEF_power_pin
     -power LEF_power_pin -ground LEF_ground_pin
     [-valid_location {to | from}]
     { -enable pin | -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 | voltage_range}
     -output_voltage_range {voltage | voltage_range}
     -direction down
     [-input_power_pin LEF_power_pin] [-output_power_pin LEF_power_pin]
     [-ground LEF_ground_pin]
     [-valid_location {to | from}]
```

Note: You cannot use the `-enable` option in the `define_level_shifter_cell` definition.In addition, you must specify the same value for the `-valid_location` option in both commands.

Tip

If you want to model an enabled level shifter, see Modeling an Enabled Level Shifter on page 155.

Modeling an Isolation Cell with Multiple Enable Pins

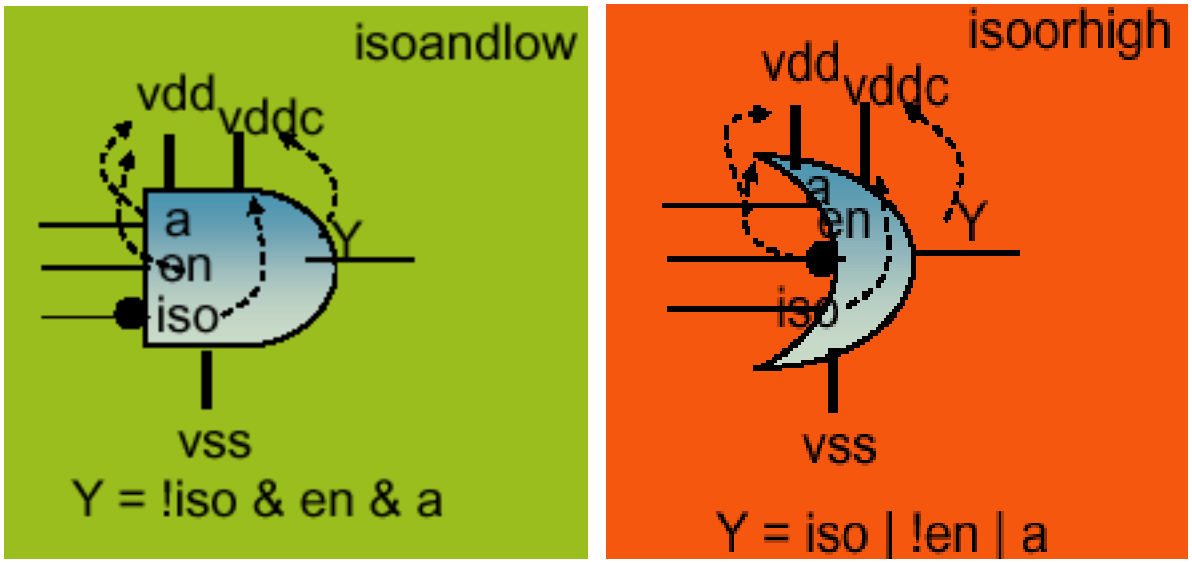
Some isolation cells have besides an enable pin that is related to the non-switchable supply of the cell, additional enable pins that are related to the switchable supply. The switchable enable pin can be used to synchronize the isolation logic right before the non-switchable enable pin is activated or deactivated. To model an isolation cell with multiple enable pins, use the `define_isolation_cell` command with the following options.

```
define_isolation_cell
     -cells cell_list [-library_set library_set]
     -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}]
```

When you specify an isolation rule that targets these types of isolation cells, you need to use the `create isolation rule` command with the `-isolation_control` option with control type `sync_enable`.

Figure 5-17shows two examples of cells with multiple enable pins. The `iso` enable pin is related to the non-switchable supply `vddc`, while the `en` enable pin is related to the switchable supply `vdd`.

Figure 5-17 Isolation Cells with Multiple Enable Pins



The following command models the `isoandlow` and `isoorhigh` cells in Figure 5-17 on page 175:

```
define_isolation_cell \
     -cells {isoandlow isoorhigh} \
     -aux_enables en \
     -power_switchable vdd \
     -power vddc -ground vss \
     -enable iso
```

The following commands show the isolation rules that target the `isoandlow` and `isoorhigh` cells in Figure 5-17 on page 175:

```
create_isolation_rule \
     -name iso1 \
     -isolation_condition iso_drvr \
     -from PD1 \
     -isolation_output low \
     -isolation_control { {sync_enable en_drvr} } \
     -secondary_domain iso_supply_domain

create_isolation_rule \
     -name iso2 \
     -isolation_condition iso_drvr \
     -from PD1 \
     -isolation_output high \
     -isolation_control { {sync_enable en_drvr} } \
     -secondary_domain iso_supply_domain
```

Modeling an Isolation Latch with a Set or Reset Pin

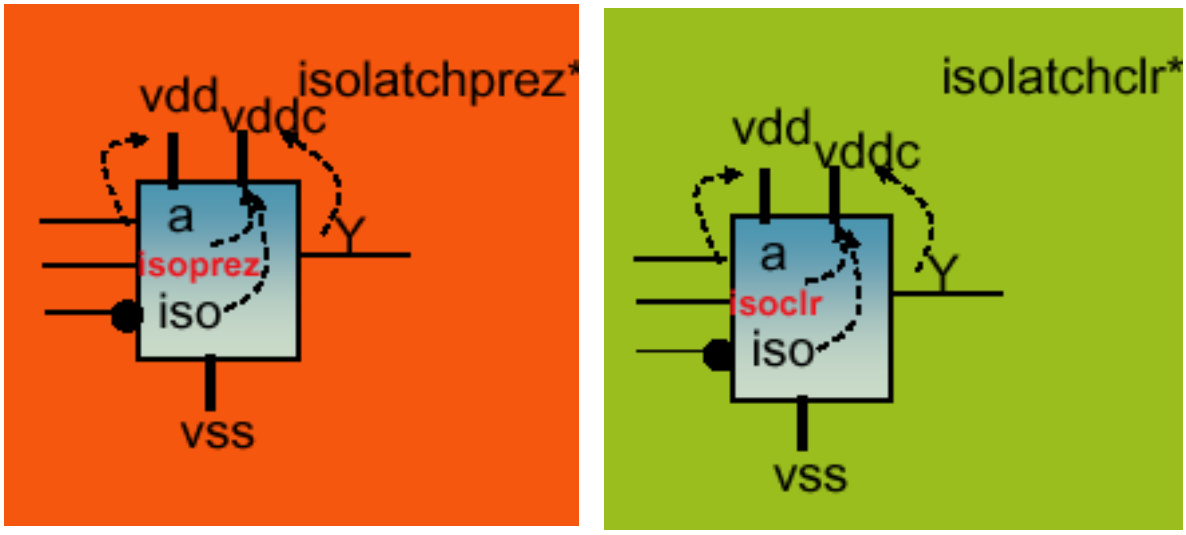
Some isolation latches are designed with a set or reset pin to ensure that the latch output has a known value at the time when the design is initially powered up. To model an isolation latch with a set or reset pin, use the `define_isolation_cell` command with the following options.

```
define_isolation_cell
     -cells cell_list [-library_set library_set]
     -always_on_pins 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}]
```

When you specify an isolation rule that targets these types of isolation cells, you need to use the `create_isolation_rule` command with the `-isolation_control` option with control type `set` or `reset` and the `-isolation_output` option with value `hold`.

Figure 5-18 shows two examples of such latches. The `iso` enable pin, `isoprez` (set), the `isoclr` (reset) pins are all related to the non-switchable supply `vddc`.

Figure 5-18 Isolation Cells with Set and Reset Pins



The following commands model the `isolatchclr` and `isolatchprez` cells in Figure 5-18 on page 177:

```
define_isolation_cell
     -cells isolatchclr
     -always_on_pins isoprez
     -power_switchable vdd
     -power vddc -ground vss
     -enable iso

define_isolation_cell
     -cells isolatchclr
     -always_on_pins isoclr
     -power_switchable vdd
     -power vddc -ground vss
     -enable iso
```

The following commands show the isolation rules that targets the `isolatchclr` and `isolatchprez` cells in Figure 5-18 on page 177:

```
create_isolation_rule \
     -name iso1 \
     -isolation_condition iso_drvr \
     -from PD1
     -isolation_output hold \
     -isolation_control { {set iso_prez} } \
     -secondary_domain iso_supply_domain

create_isolation_rule \
     -name iso2 \
     -isolation_condition iso_drvr \
     -from PD1 \
     -isolation_output hold \
     -isolation_control { {reset iso_clr} } \
     -secondary_domain iso_supply_domain
```

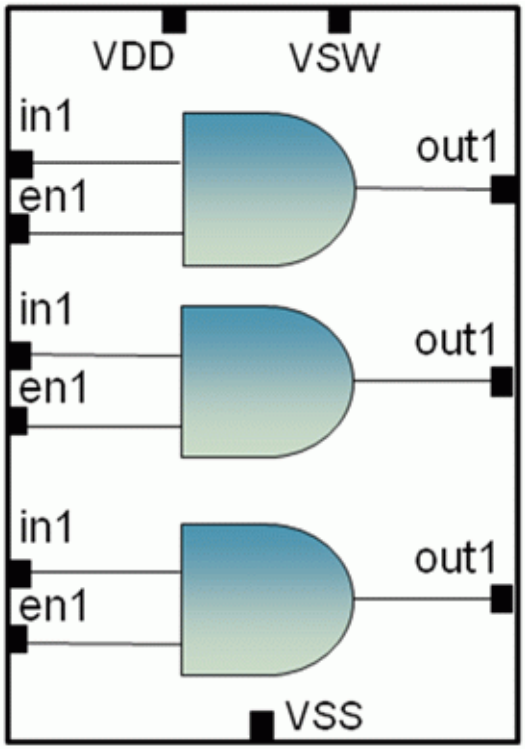
Modeling a Multi-bit Isolation Cell

A multi-bit isolation cell has multiple pairs of input and output pins with each pair serving as a single bit isolation cell. An example is shown in Figure 5-19.

If the cell uses the same enable pin for all pairs of input and output pins, there is no difference in modeling such a multi-bit cell with respect to the single bit isolation cell.

If the cell has different enable pins for the input and output pairs, you must model the cell using the `-pin_groups` option of the `define_isolation_cell` command.

Figure 5-19 Multi-bit Isolation Cell



The following CPF command can be used to describe the multi-bit isolation cell for power switchable domain shown in Figure 5-19 (see Figure 5-13 on page 169 for the corresponding single bit cell).

```
define_isolation_cell -cells IsoLL \
-power_switchable VSW \
-power VDD -ground VSS \
-pin_groups {{in1 out1 en1} {in2 out2 en2} {in3 out3 en3}}
```

Modeling a Complex Isolation Cell

For complex isolation cells that cannot be modeled using the previous techniques, CPF provides a generic mechanism.

To model a complex isolation cell, use the `update_isolation_rules` command with the following options.

```
update_isolation_rules -names rule_list
     -cells cell_list [-use_model -pin_mapping pin_mapping_list
          [-domain_mapping domain_mapping_list] ]
```

In this case, the first cell specified in the cell list refers to the complicated isolation cell that cannot be modelled using the `define_isolation_cell` command. The cell name that you specify must correspond to both the model name used in simulation and the cell name used for implementation.

Use the following format for the *`pin_mapping_list`* to specify the connection of each cell pin to a pin or port in the design:

{*cell_pin design_pin_reference*}

- *`cell_pin`* is the name of the pin in the cell definition
- *`design_pin_reference`* is one of the following:
  - the name of a design port or instance pin—You can prepend the "!" character to the name if the inverse of the port/pin is used to drive the cell pin
  - *`isolation_signal`*, which refers to the expression in `-isolation_condition`

If the cell has a corresponding macro model definition in CPF, you can specify the mapping of the domains in the macro model to the top-level domains.

Use the following format to specify a domain mapping:

{*domain_in_child_scope domain_in_parent_level_scope*}

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

If the macro model has multiple power domains defined, this option must be used.

Modeling State Retention Cells

Types of State Retention Cells

State retention cells are used for sequential cells to keep their previous state prior to power down.

CPF can describe different types of state retention cells. The following is a list of the most typical state retention cells:

- State Retention Cell with Save Control
- State Retention Cell with Restore Control
- State Retention Cells with Save and Restore Controls
- State Retention Cells without Save or Restore Control

All types of state retention cells are defined using the `define_state_retention_cell` command. The following sections indicate which command options to use for each type.

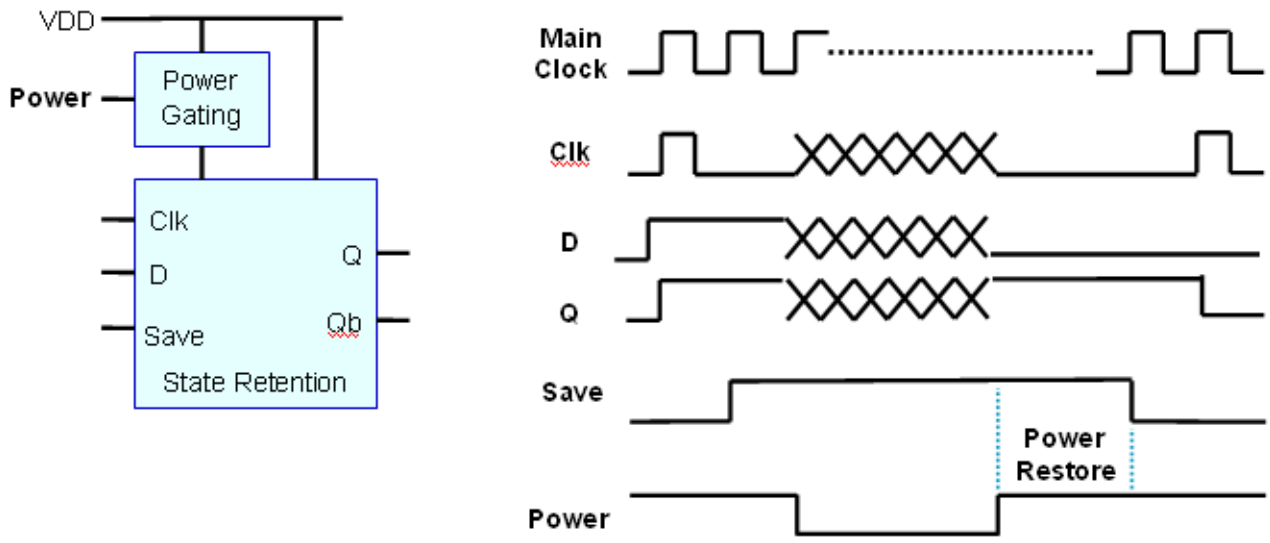
State Retention Cell that Restores when Power is Turned On

To model a state retention cell that saves the current value when the control pin becomes active while the power is on, retains the saved value when power is off, and restores the saved value when the power is turned on, use the following command options:

```
define_state_retention_cell
    -cells cell_list [-library_set library_set]
    [-cell_type string]
    [-always_on_pins pin_list]
    [-clock_pin pin]
    -save_function expression
    [-restore_check expression] [-save_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 ]
```

Figure 5-20 shows an example of such a cell.

Figure 5-20 State Retention with Save Control



To model the cell shown in Figure 5-20, use the following command:

```
define_state_retention_cell -cells SR1 \
    -clock_pin Clk \
    -save_function save \
    -restore_check !Clk -save_check !Clk \
    -power_switchable VSW \
    -power VDD -ground VSS
```

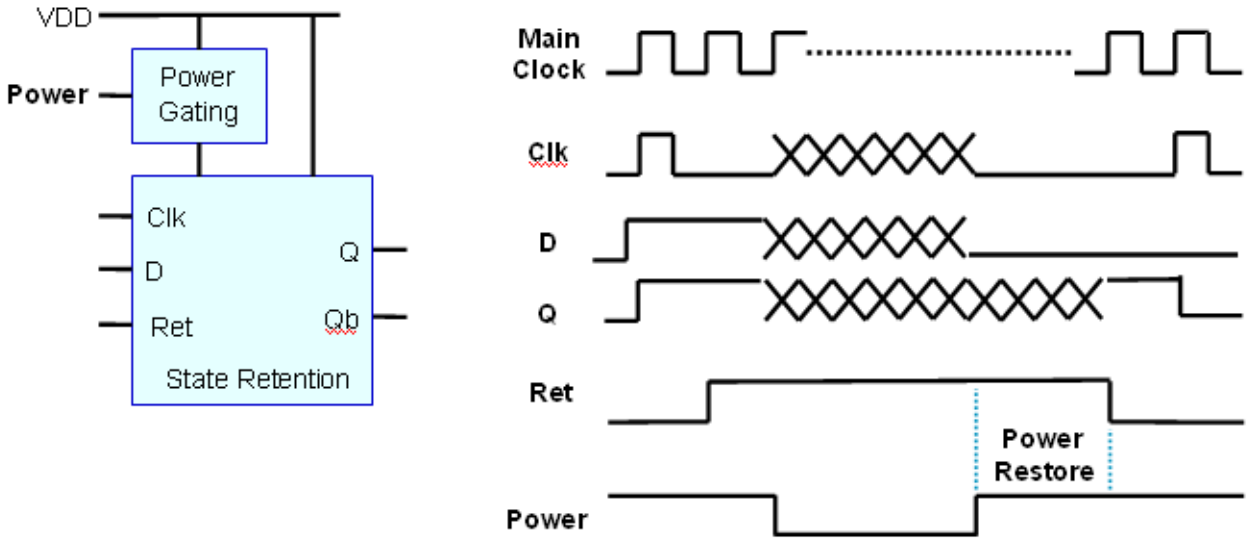
State Retention Cell that Restores when Control Signal is Deactivated

To model a state retention cell that saves the current value when the control pin becomes deactivated and restores the saved value when the control signal becomes activated, use the following command options:

```
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
    [-restore_check expression] [-save_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 ]
```

Figure 5-21 shows an example of such a cell.

Figure 5-21 State Retention with Restore Control



To model the cell shown in Figure 5-21, use the following command:

```
define_state_retention_cell -cells SR1 \
    -clock_pin Clk \
    -restore_function !Ret \
    -restore_check !Clk -save_check !Clk \
    -power_switchable VSW \
    -power VDD -ground VSS
```

State Retention Cells with Save and Restore Controls

If you have a state retention cell with both save and restore control, the cell saves the current value when the save control pin is activated and the power is on, while the cell restores the saved value when the restore control pins is activated. To model such a cell use the following command options:

```
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_check expression] [-save_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 ]
```

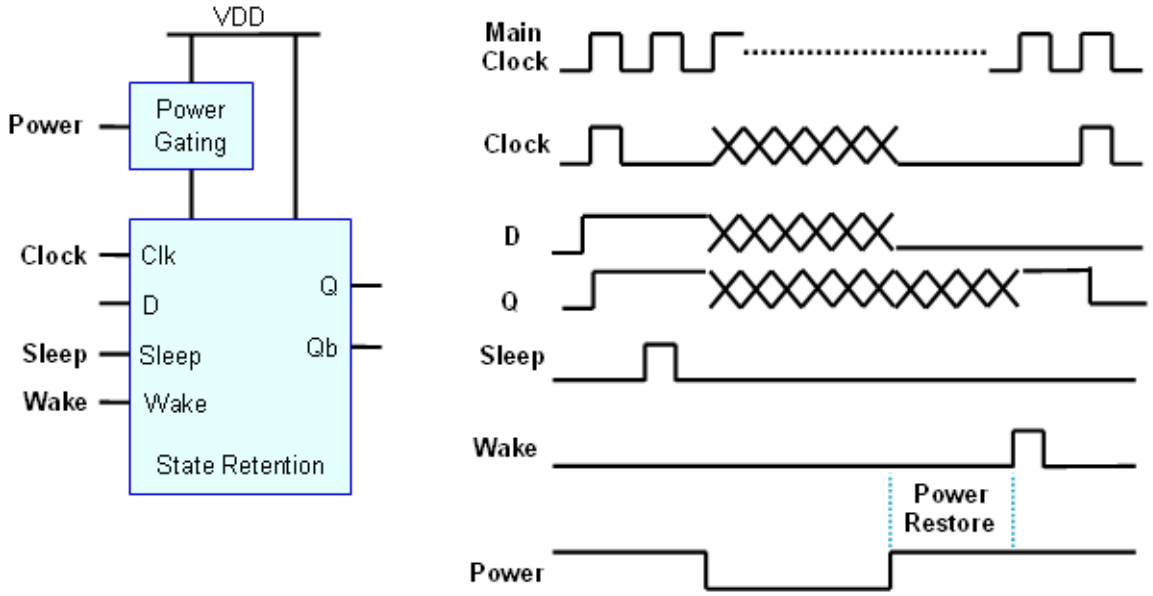
In this case, the cell saves the current value when the save expression is `true` and the power is on. The cell restores the saved value when the restore expression is `true` and the power is on.

It is an error if you use the same pin or same expressions for both the save and restore function. For example the following two commands are not correct:

```
define_state_retention_cell -cells My_Cell -restore_function pg -save_function !pg
define_state_retention_cell -cells foo -restore_function pg -save_function pg
```

Figure 5-22 on page 185 shows an example of such a cell.

Figure 5-22 State Retention with Save and Restore Controls



To model the cell shown in Figure 5-22, use the following command:

```
define_state_retention_cell -cells SR2 \
    -clock_pin Clk \
    -restore_function Wake -save_function Sleep \
    -restore_check "!Clk" -save_check "!Clk" \
    -power_switchable VSW \
    -power VDD -ground VSS
```

The state will be saved when Sleep is active and the clock is down and the state will also be restored when Wake is active and the clock is down.

State Retention Cells without Save or Restore Control

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

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

To model such a cell use the following command options:

```
define_state_retention_cell
    -cells cell_list [-library_set library_set]
    [-cell_type string]
    [-always_on_pins pin_list]
    [-clock_pin pin]
    -save_function expression
    [-restore_check expression] [-save_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 ]
```

When you specify a state retention rule that targets these types of state retention cells, you need to use the create_state_retention_rule command with the -save_edge option and the clock signal in the expression.

The following example models master-slave retention cell ms_ret:

```
define_state_retention_cell -cells ms_ret \
    -clock_pin CLK \
    -save_function CLK \
    -restore_check "!CLK" -save_check "!CLK"
```

The following command shows the state retention rule that targets cell ms_ret

```
create_state_retention_rule -name sr1 \
    -domain PD1 \
     -save_edge clock
```

Modeling a Complex State Retention Cell

For complex state retention cells that cannot be modeled using the previous techniques, CPF provides a generic mechanism. To model such a cell, use the `update_isolation_rules` command with the following options.

```
update_state_retention_rules -names rule_list
     -cells cell_list [-use_model -pin_mapping pin_mapping_list
          [-domain_mapping domain_mapping_list] ]
```

In this case, the first cell specified in the cell list refers to the complicated state retention cell that cannot be modelled using the `define_state_retention_cell` command. The cell name that you specify must correspond to both the model name used in simulation and the cell name used for implementation.

Use the following format for the *`pin_mapping_list`* to specify the connection of each cell pin to a pin or port in the design:

{*`cell_pin design_pin_reference`*}

- *`cell_pin`* is the name of the pin in the cell definition
- *`design_pin_reference`* is one of the following:
  - the name of a design port or instance pin. You can prepend the "!" character to the name if the inverse of the port/pin is used to drive the cell pin
  - *`save_signal`*, which refers to the expression in `-save_edge` or `-save_level` option
  - *`restore_signal`*, which refers to the expression in `-restore_edge` or `-restore_level` option

If the cell has a corresponding macro model definition in CPF, you can specify the mapping of the domains in the macro model to the top-level domains.

Use the following format to specify a domain mapping:

{*`domain_in_child_scope domain_in_parent_level_scope`*}

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

If the macro model has multiple power domains defined, this option must be used.

Modeling Power Switch Cells

Types of Power Switch Cells

To connect and disconnect the power (or ground) supply from the gates in internal switchable power domains, you must add power switch logic.

CPF can describe different types of power switch cells. The following is a list of the most typical cells:

- single stage power switch cell—single transistor that controls the primary power supply to the logic of an internal switchable domain
- single stage ground switch cell—single transistor that controls the primary ground supply to the logic of an internal switchable domain
- dual-stage power switch—power switch with a weak and strong transistor to control the primary power supply to the logic of an internal switchable domain
- dual-stage ground switch—ground switch with a weak and strong transistor to control the primary ground supply to the logic of an internal switchable domain

All types of power switch cells are defined using the `define_power_switch_cell` command. The following sections indicate which command options to use for each type.

Modeling a Single Stage Power Switch Cell

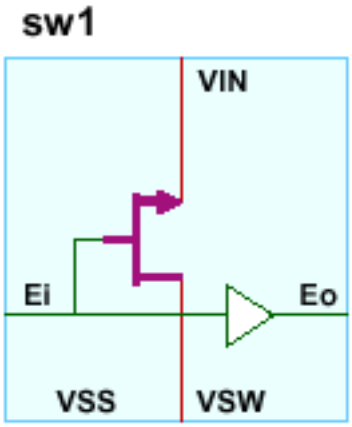
To model a single stage power switchable power switch cell use the `define_power_switch_cell` command with the following options:

```
define_power_switch_cell
     -cells cell_list [-library_set library_set]
     -stage_1_enable expression [-stage_1_output expression]
     -type header
     -power_switchable LEF_power_pin  -power LEF_power_pin
     [-ground LEF_ground_pin]
     [-enable_pin_bias [float:]float] [ -gate_bias_pin LEF_power_pin]
     [ -stage_1_on_resistance float ] [ -stage_1_saturation_current float]
     [ -leakage_current float ]
```

In case of an unbuffered power switch cell, you do not need to specify the `-stage_1_output` and `-ground` options.

Figure 5-23 shows a power switch cell with an internal buffer. `VIN` is the pin connected to the unswitched power. `VSW` is the pin connected to the switchable power that is connected to the logic. When the enable signal `Ei` is activated the unswitched power is supplied to the logic. As shown in Figure 5-23, this type of cell usually contains a buffer that allows multiple power switch cells to be chained together to form a power switch column or ring. However, the power and ground of this buffer must be unswitchable.

Figure 5-23 Single Stage Power Switch



The following command models the power switch cell shown in Figure 5-23:

```
define_power_switch_cell \
     -cells sw1 \
     -stage_1_enable Ei -stage_1_output Eo \
     -type header \
     -power_switchable VSW -power VIN -ground VSS
```

Modeling a Power Switch cell with Gate Bias

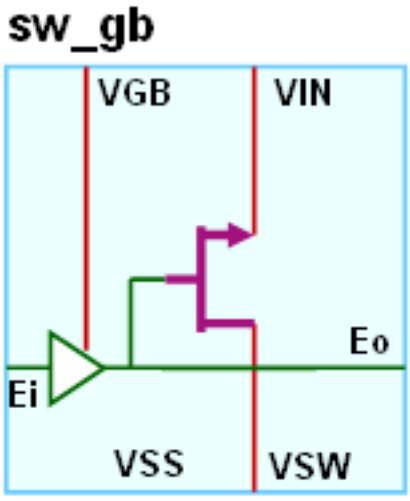
To model a single stage power switchable power switch cell with gate bias use the `define_power_switch_cell` command with the following options:

```
define_power_switch_cell
     -cells cell_list [-library_set library_set]
     -stage_1_enable expression [-stage_1_output expression]
     -type header
     -power_switchable LEF_power_pin  -power LEF_power_pin
          [-ground LEF_ground_pin]
     -enable_pin_bias [float:]float -gate_bias_pin LEF_power_pin
     [ -stage_1_on_resistance float ] [ -stage_1_saturation_current float]
     [ -leakage_current float ]
```

Typically the enable pin is related to the power and the ground pin. With gate bias the enable pin is related to the gate bias pin and the ground. The voltage on the gate bias pin is larger than the voltage of the power pin. Such as cell creates less leakage power compared to the cell without gate bias.

In Figure 5-24, the gate bias pin is `VGB`. Assume that input voltage `VIN` is at 1.2V and the gate bias pin is at 3.3 V.

Figure 5-24 Single Stage Power Switch with Gate Bias



The following command models the power switch cell shown in Figure 5-24:

```
define_power_switch_cell \
     -cells sw1 \
     -stage_1_enable Ei -stage_1_output Eo \
     -type header \
     -power_switchable VSW -power VIN -ground VSS \
     -gate_bias_pin VGB -enable_pin_bias 2.1
```

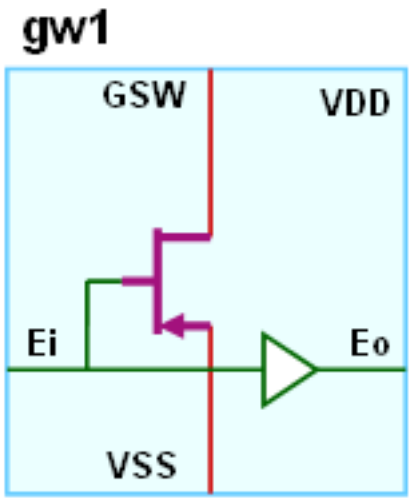
Modeling a Single Stage Ground Switch Cell

To model a single stage ground switchable power switch cell use the `define_power_switch_cell` command with the following options:

```
define_power_switch_cell
     -cells cell_list [-library_set library_set]
     -stage_1_enable expression [-stage_1_output expression]
     -type footer
     -ground_switchable LEF_ground_pin  -ground LEF_ground_pin
           [-power LEF_power_pin]
     [-enable_pin_bias [float:]float] [ -gate_bias_pin LEF_power_pin]
     [ -stage_1_on_resistance float ] [ -stage_1_saturation_current float]
     [ -leakage_current float ]
```

Figure 5-25 shows a ground switch cell. `VSS` is the pin connected to the unswitched ground. `VSW` is the pin connected to the switchable ground that is connected to the logic. When the enable signal `Ei` is activated the unswitched ground is supplied to the logic. As shown in Figure 5-25, this type of cell usually contains a buffer that allows multiple ground switch cells to be chained together to form a ground switch column or ring. However the power and ground of this buffer must be unswitchable.

Figure 5-25 Single Stage Ground Switch



The following command models the ground switch cell shown in Figure 5-25:

```
define_power_switch_cell \
     -cells gw1 \
     -stage_1_enable Ei -stage_1_output Eo \
     -type header \
     -ground_switchable GSW -ground VSS -power VDD
```

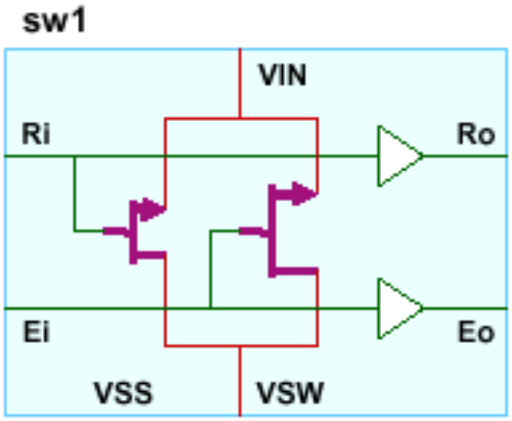
Modeling a Dual-Stage Power Switch Cell

To model a power switch cell with two stages, use the following command options:

```
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 header
     -power_switchable LEF_power_pin  -power LEF_power_pin
          [-ground LEF_ground_pin]
     [-enable_pin_bias [float:]float] [ -gate_bias_pin LEF_power_pin]
     [ -stage_1_on_resistance float ] [ -stage_1_saturation_current float]
     [ -stage_2_on_resistance float ] [ -stage_2_saturation_current float]
     [ -leakage_current float ]
```

Figure 5-26 shows a dual-stage power switch cell. `VIN` is the pin connected to the unswitched power. `VSW` is the pin connected to the switchable power that is connected to the logic. Only when both enable signals Ri and `Ei` are activated can the unswitched power be supplied to the logic. The `Ri` enable signal drives the stage-1 (weak) transistor which requires less current to restore the unswitched power. The `Ei` enable signal drives the stage-2 (strong) transistor which requires more current to fully supply the unswitched power to the logic. This type of cell usually contains two buffers that allow multiple power switch cells to be chained together to form a power switch column or ring. However the power and ground of these buffers must be unswitchable.

Figure 5-26 Dual-Stage Power Switch



The following command models the power switch cell shown in Figure 5-26:

```
define_power_switch_cell \
     -cells sw1 \
     -stage_1_enable Ri -stage_1_output Ro \
     -stage_2_enable Ei -stage_2_output Eo \
     -type header \
     -power_switchable VSW -power VIN -ground VSS
```

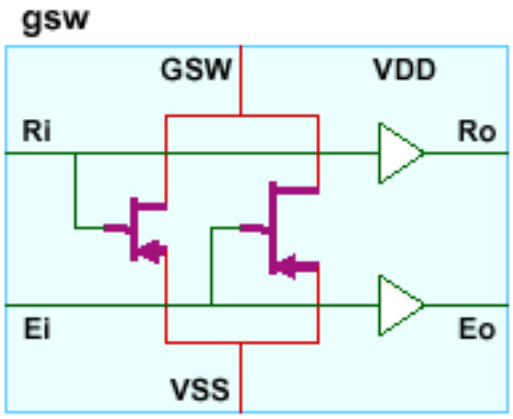
Modeling a Dual-Stage Ground Switch Cell

To model a ground switch cell with two stages, use the following command options:

```
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
     -ground_switchable LEF_ground_pin  -ground LEF_ground_pin
          [-power LEF_power_pin]
     [-enable_pin_bias [float:]float] [ -gate_bias_pin LEF_power_pin]
     [ -stage_1_on_resistance float ] [ -stage_1_saturation_current float]
     [ -stage_2_on_resistance float ] [ -stage_2_saturation_current float]
     [ -leakage_current float ]
```

Figure 5-27 shows a dual-stage ground switch cell. `VSS` is the pin connected to the unswitched ground. `GSW` is the pin connected to the switchable ground that is connected to the logic. Only when both enable signals Ri and `Ei` are activated can the unswitched ground be supplied to the logic. The `Ri` enable signal drives the stage-1 (weak) transistor which requires less current to restore the unswitched ground. The `Ei` enable signal drives the stage-2 (strong) transistor which requires more current to fully supply the unswitched ground to the logic. This type of cell usually contains two buffers that allow multiple ground switch cells to be chained together to form a ground switch column or ring. However the power and ground of these buffers must be unswitchable.

Figure 5-27 Dual-Stage Ground Switch



The following command models the ground switch cell shown in Figure 5-27:

```
define_power_switch_cell \
     -cells gsw \
     -stage_1_enable Ri -stage_1_output Ro \
     -stage_2_enable Ei -stage_2_output Eo \
     -type footer \
     -ground_switchable GSW -ground VSS -power VDD
```

Modeling Pad Cells

CPF provides two methods to model a cell pad:

- Using a Pad Cell Definition to Create a Simplified Pad Cell Model.
- Using a CPF Macro Model to Create a Detailed Pad Cell Model

  The detailed model is required when your pad cell contains any of the following:

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

Note: if you created a simplified and detailed model, the detailed model (macro model definition) will take precedence.

Every instance of a pad cell must be instantiated in CPF:

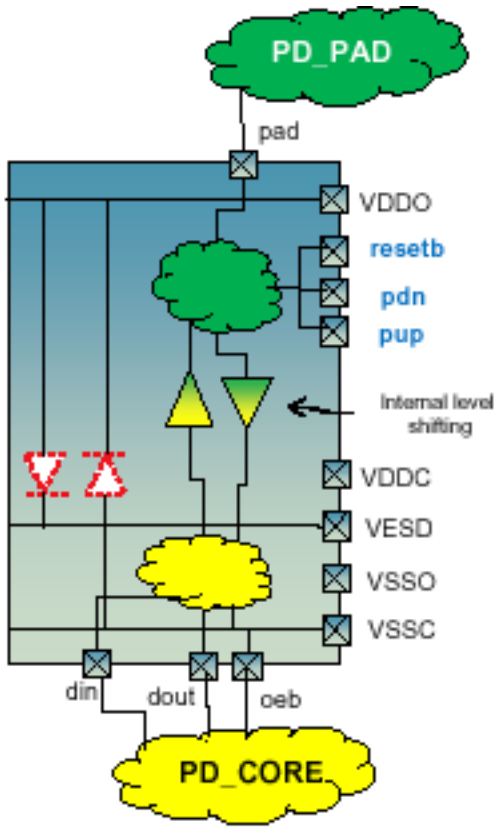
- If you created a simplified model, you must declare the pad instances with the `create_pad_rule` command.
- If you created a detailed model, you can instantiate the pad instances with the `set_instance` command or with the `create_pad_rule` command.

An instance is identified as a pad instance if the cell is defined in the `define_pad_cell` or there is a macro model definition for the cell that has the `set_pad_ports` command included.

Using a Pad Cell Definition to Create a Simplified Pad Cell Model

Consider the pad shown in Figure 5-28 on page 198. This I/O pad cell has one pin (`pad`) that will be directly connected to the board. The signal coming in through this pin goes through internal level shifting that produces three signals connected to the core of the chip. The input pins `pad`, `resetb`, `pdn`, and `pup` connect to logic whose power is supplied by `VDDO` and `VSSO`, while the three other output pins (`din`, `dout`, and `oeb`) are driven by logic whose power is supplied by `VDDC` and `VSSC`.

Figure 5-28 I/O Pad Cell



For a simplified pad cell model you need to declare

- The pins that connect directly to the package or the board
- The groups of pins that are related to the same power supplies

  You can specify a power or ground pin in multiple pin groups.

  Each power and ground pin must belong to at least one group.

If a group has more than one power (ground) pin, these power (ground) pins are considered to be equivalent for that group.

If a power or ground pin appears in multiple pin groups, the top-level power domains that those groups are mapped to must have the same primary power or ground net defined. Otherwise, it is an error.

Pins in a group without power and ground pins, or not specified in any pin group are considered to be floating pins.

It is an error if you specify a non-power or ground pin in multiple pin groups.

- The pins which connect to internal isolation logic

This information can be given with the `define_pad_cell` command as shown in Figure 5-29.

When you defined a simple pad cell model, you need to declare the pad instances using the `create_pad_rule` command. The pad rule defines how to map the pin groups of the pad instances to the top-level power domains.

Important

It is an error If a pin group of a pad cell is not specified in the mapping option.

Figure 5-29 on page 200 shows an extract of a CPF file with the simplified pad cell definition and pad instantiation for the pad shown in Figure 5-28 on page 198.

In this example all instantiations of cell `data_io_3V` that are connected to the ports of `FOO[0:127]` follow the same domain mapping specified in the pad rule. The pins `resetb`, `pdn`, `pup`, and `pad` belong to domain `PD_PAD` and the power pin `VDDO` and the ground pin `VSSO` must be connected to the primary power and ground nets of domain `PD_PAD`, respectively.

Similarly, the pins `din`, `dout`, and `oeb` belong to domain `PD_CORE` and the power pin `VDDC` and the ground pin `VSSC` must be connected to the primary power and ground nets of domain `PD_CORE`, respectively.

Figure 5-29 Pad Cell Definition and Instantiation

```
define_pad_cells -cells data_io_3V  \
    -pad_pins pad \
    -pin_groups { IO:{ VDDO VSSO resetb pdn pup pad} \
                  CORE:{ VDDC VSSC din dout oeb} }

set_design top

create_power_domain -name PD_PAD ...

create_power_domain -name PD_CORE ...

create_pad_rule -name pad1 -of_bond_ports {FOO[0:127]}  \
    -mapping { {IO PD_PAD} {CORE PD_CORE}}
...
end_design
```

Using a CPF Macro Model to Create a Detailed Pad Cell Model

For the detailed model, you must declare the pins that connect directly to the package or the board in the macro model. Use the set_pad_ports command (between the set_macro_model and end_macro_model commands) to do this.

To model internal diodes in the macro model, you can use the set_diode_ports command to declare the pins of the macro cell that connect to the positive and negative pins of a diode.

To declare the instances of a pad cell in the CPF file, you can use either the set_instance command or the create_pad_rule command. The pad rule defines how to map the power domains of the pad cell macro model to the top-level power domains.

Figure 5-29 on page 200 shows an extract of a CPF file with the detailed pad cell definition and pad instantiation for the pad shown in Figure 5-28 on page 198.

The semantics of this example is the same as for the example shown in Figure 5-29 on page 200.

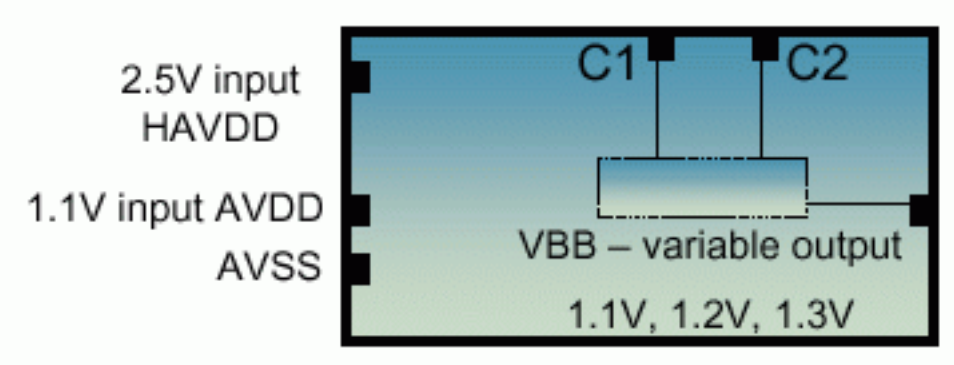
Figure 5-30 Detailed Pad Cell Model and Instantiation

```
set_macro_model data_io_3V
set_pad_ports { pad }
create_power_domain  -name PDCore -default \
    -boundary_ ports  {din dout oeb}
update_power_domain -name PDCore -primary_power_net VDDC -primary_ground_net VSSC
create_power_domain  -name PDIO  -boundary_ ports  {VDDO VSSO resetb pdn pup pad}
update_power_domain -name PDIO -primary_power_net VDDO -primary_ground_net VSSO
end_macro_model
set_design top
create_power_domain  -name PD_PAD ...
create_power_domain  -name PD_CORE ...
create_pad_rule  -name -of_bond_ports {FOO[ 0: 127]} \
    -mapping { {PDIO PD_PAD} {PDCore PD_CORE}}
...
end_design
```

Modeling a Voltage Regulator

Figure 5-31 on page 202 shows an example of a voltage regulator. Pin `HAVDD` is the pin to which the high voltage supply needed for the proper operation of the voltage regulator is applied. The reference power and ground supplies are applied to pins `AVDD` and `AVSS`, respectively. The control signals to regulate the voltage are applied to pins `C1` and `C2`. Pin `VBB` corresponds to the output power supply.

Figure 5-31 Voltage Regulator



The voltage regulator of Figure 5-31 has three power domains. The power domain associated with the output power supply is referred to as a *power source domain*.

To model a voltage regulator, you need to use a CPF macro model:

- The power source domain must be identified with the `-power_source` option in the `create_power_domain`.
- The power domain of the high input voltage supply must be declared as the base domain (`-base_domains`) of the power source domain.

  Note: The primary supply voltage of the base domain can be different from the voltage of the power source domain.
- The options of the `update_power_domain` command for the power source domain depend on whether the output power supply is used as power source for the primary power supply or body bias supply of other domains:
  - Use the `-primary_power_net` option if the power source can only be used a primary power supply of other domains
  - Use the `-primary_ground_net` option if the power source is a primary ground supply of other domains

    This occurs when a top design uses the ground bias supply to control the power

dissipation and performance and uses the output of the voltage regulator for the ground connection.

- Use the `-pmos_bias_net` or `-nmos_bias_net` option (without the `-primary_power_net` or `-primary_ground_net` specification) if the power source can only be used as body bias supply of other domains
- Use the `-primary_power_net` and `-pmos_bias_net` options if the power source can be used as a primary power supply and pmos body bias supply of other domains
- Use the `-primary_ground_net` and `-nmos_bias_net` options if the power source can be used as a primary ground supply and nmos body bias supply of other domains

Note: If the macro has an internal regulator whose output supply only drives logic within the macro, the `update_power_domain` command for the power source domain will not have any primary power, primary ground, or body bias net specifications. The power source domain is then referred to as an *internal power source domain*.

- Specify the reference signal for the power source domain with the `set_power_source_reference_pin` command.
- In any power mode definition, the nominal condition for the power source domain should have a lower voltage range than the nominal condition for its base domain.

Examples 5-1 through 5-3 apply to Figure 5-31 on page 202.

Example 5-1 Voltage Regulator Model for Output Power Supply Used as PMOS Body Bias Supply

Power supply `VBB` will be used as the PMOS body bias supply of top-level domain `PDDVFS`.

```
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_power_domain -name PDREF -boundary_ports {C1 C2}
update_power_domain -name PDREF -primary_power_net AVDD -primary_ground_net AVSS
...
set_power_source_reference_pin AVDD -domain PDVOUT voltage_range 1.0:1.1
end_macro_model regulator

set_design top
create_power_domain -name PDDefault -default
update_power_domain -name PDDefault -primary_power_net AVDD \
    -primary_ground_net AVSS
create_power_domain -name PDDirty
update_power_domain -name PDDirty -primary_power_net HighVdd \
    -primary_ground_net VSS
create_power_domain -name PDDVFS -instances ...
update_power_domain -name PDDVFS -primary_power_net VDD -primary_ground_net VSS \
```

```
    -pmos_bias_net VBB -nmos_bias_net VSS0
set_instance I_regulator -domain_mapping { {PDVIN PDDirty} {PDREF PDDefault} \
    {PDVOUT PDDVFS}}
....
```

Example 5-2 Voltage Regulator Model for Output Power Supply Used as PMOS Body Bias Supply with Voltage Range Specification

In this example, the CPF contains voltage ranges for the nominal condition of the power source domain but does not specify how the output voltage is controlled. The simulation model could contain the output voltage behavior.

```
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_power_domain -name PDREF -boundary_ports {C1 C2}
update_power_domain -name PDREF -primary_power_net AVDD -primary_ground_net AVSS
create_nominal_condition -name LDO_range -voltage 1.1 -pmos_bias_voltage 1.1:1.3
create_nominal_condition -name HVDD -voltage 2.45:2.55 -ground_voltage 0.0
create_nominal_condition -name REF -voltage 1.1 -ground_voltage 0.0
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 1.1:1.1
end_macro_model regulator
```

Example 5-3 Voltage Regulator Model for Output Power Supply Used as PMOS Body Bias Supply with Voltage Control Specification

In this example, the CPF does specify how the output voltage is controlled.

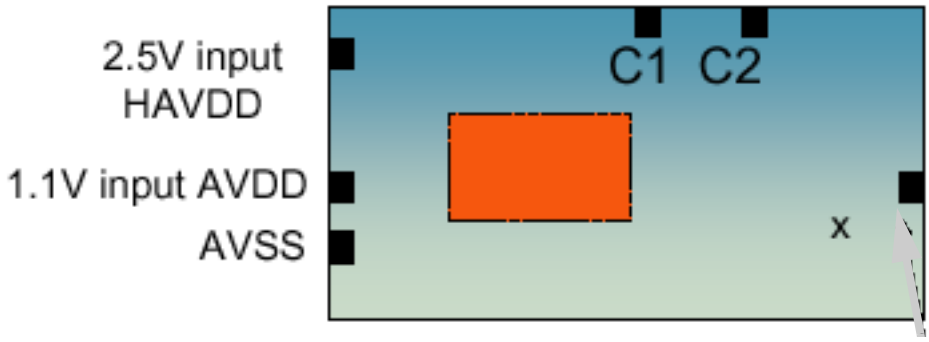
VBB=1.1v when C1=1, C2=0; VBB=1. 2v when C1=0, C2=1; VBB=1. 3v when C1=1, C2=1

```
set_macro_model regulator
create_nominal_condition -name LDO1 -voltage 1.1 -pmos_bias_voltage 1.1
create_nominal_condition -name LDO2 -voltage 1.1 -pmos_bias_voltage 1.2
create_nominal_condition -name LDO3 -voltage 1.1 -pmos_bias_voltage 1.3
create_nominal_condition -name HVDD -voltage 2.5 -ground_voltage 0.0
create_nominal_condition -name REF -voltage 1.1 -ground_voltage 0.0
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 \
    -active_state_conditions { LDO1@"C1&!C2" LDO2@"!C1&C2" LDO3@"C1&C2" }
update_power_domain -name PDVOUT -pmos_bias_net VBB
create_power_domain -name PDREF -boundary_ports {C1 C2}
update_power_domain -name PDREF -primary_power_net AVDD -primary_ground_net AVSS
create_power_mode -name PM1 -default -domain_conditions \
    { PDREF@REF PDVIN@HVDD PDVOUT@LDO1 }
create_power_mode -name PM2 -domain_conditions \
    { PDREF@REF PDVIN@HVDD PDVOUT@LDO2 }
create_power_mode -name PM3 -domain_conditions \
    { PDREF@REF PDVIN@HVDD PDVOUT@LDO3 }
set_power_source_reference_pin AVDD -domain PDVOUT voltage_range 1.1:1.1
end_macro_model regulator
```

Example 5-4 Macro Cell with an Internal Regulator

The macro below has an internal power regulator. The macro has no pin to export the internal power supply. Some output data pins are driven by logic that are in tern driven by the internal supply.

Figure 5-32 Internal Voltage Regulator




```
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
create_power_domain -name PDREF -boundary_ports {C1 C2}
update_power_domain -name PDREF -primary_power_net AVDD -primary_ground_net AVSS
create_nominal_condition -name OUT -voltage 0.9:1.1
create_nominal_condition -name HDVV -voltage 2.5 -ground_voltage 0.0
create_nominal_condition -name REF -voltage 1.1 -ground_voltage 0.0

create_power_mode -name PM -default -domain_conditions \
    { PDREF@REF PDVIN@HVDD PDVOUT@OUT }
set_power_source_reference_pin AVDD -domain PDVOUT voltage_range 1.1:1.1
end_macro_model regulator
```

Power source domain PDVOUT has no `update_power_domain` command that contains any primary power, primary ground, or body bias net specifications.

Power Domain Mapping of a Power Source Domain

There are special semantics for power domain mapping in case of a power source domain,

The top-level domain inherits all the attributes related to the macro model power source domain:

- The top-level domain uses the power ports of a power source domain as either primary supply or body bias supply depending on the specification in the macro model
- If the top-level domain already has a shutoff condition specified, it must be functionally equivalent to the conditions specified for the power source domain and the domain must be an external switchable domain.
- If the top-level domain also has active state conditions specified, it must be functionally equivalent to the corresponding condition specified for the power source domain.

A power source domain can be mapped into multiple top-level domains.

- These top-level domains must have consistent nominal conditions in all power modes
- It is an error if multiple power source domains are mapped into the same top-level domain.

The top-level domain's voltage range (specified across all power modes) must be within the voltage range specified with the power source domain.

- Depending on the definition and usage of the regulator output port, the top-level domain's nominal operating voltages can be any of the following: power voltage, ground voltage, pmos bias voltage, or nmos bias voltage, or a combination of these. For example, if a regulator output port is defined as a pmos bias supply, then among all power mode definitions for the corresponding top level domain, its pmos bias voltage must be within the range of the operating voltage of the corresponding power source domain.
- If a power source domain is switchable, and if the top-level domain it is mapped to is an unswitched domain, the top-level domain becomes switchable with the shutoff condition specified in the macro model using the cell pins. It is an error if a switchable power source domain is mapped into a top-level internal switchable domain.

If a power domain is an *internal* power source domain, the domain cannot be mapped. The `update_power_domain` command for the power source domain will not have any primary power, primary ground, or body bias port specifications.

Simulation Semantic

In simulation a power source domain is considered to be corrupted if

- Its base domain is off or operates out of the required voltage ranges
- Its reference pin is corrupted or operates out of the specified voltage ranges

When a power source domain is corrupted, the power domains mapped to this power source domain are corrupted as well. The corruption semantics applies recursively to all domain mappings that root from the power source domain.