Synopsys® Multivoltage Flow User Guide

Version December 2019, December 2019

SYNOPSYS®

Copyright and Proprietary Information Notice

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://www.synopsys.com/company/legal/trademarks-brands.html.
All other product or company names may be trademarks of their respective owners.

Free and Open-Source Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

www.synopsys.com

Feedback

Contents

Preface

This preface includes the following sections:

About This User Guide

This user guide describes the flow of the low-power solution, including synthesis, implementation, and verification. It serves as a guide to the flow and IEEE 1801 (UPF) usage in various Synopsys tools. Detailed information about the standard can be found in the IEEE 1801 specification itself, which is available from IEEE Xplore at http://ieeexplore.ieee.org. Detailed information about Synopsys product support for the multivoltage flow is available in the product manuals for individual Synopsys tools.

An important part of the low-power flow is the support for the IEEE 1801 Standard for Design and Verification of Low Power Integrated Circuits, also known as the Unified Power Format (UPF). Many Synopsys products support IEEE 1801 (UPF) infrastructure and commands, including Power Compiler, VCS LP, VC LP, Formality, IC Compiler, IC Compiler II, Fusion Compiler, PrimeTime, and PrimePower.

This user guide in intended for the following product versions:

Product	Version
Formality, IC Compiler, IC Compiler II, Fusion Compiler, Power Compiler, PrimePower, PrimeTime	Q-2019.12
VC LP	O-2018.09-SP2
VCS NLP	O-2018.09-SP2

Audience

This manual is intended for engineers who design integrated circuits using Galaxy Design Platform, and Discovery Verification Platform, and who employ reduced-power design techniques such as multivoltage power domains, power-down domains, and multiple-threshold cells.

Feedback

Related Publications

For additional information about Synopsys tools, see the documentation on the Synopsys SolvNet® online support site at the following address:

https://solvnet.synopsys.com/DocsOnWeb

You might also want to see the documentation for the following related Synopsys products:

- Design Compiler® and Power Compiler™
- DC Explorer
- IC Compiler™
- IC Compiler II
- Fusion Compiler™
- PrimeTime® and PrimePower
- Formality®
- PrimeRail
- VCS® NLP and VC LP

Release Notes

Information about new features, enhancements, changes, known limitations, and resolved Synopsys Technical Action Requests (STARs) is available in the individual product release notes on the SolvNet site.

To see the release notes for a product,

1. Go to the SolvNet Download Center located at the following address:

   https://solvnet.synopsys.com/DownloadCenter

2. Select the Synopsys product name and then select a release in the list that appears.

Feedback

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
`Courier`	Indicates syntax, such as `write_file`.
`Courier italic`	Indicates a user-defined value in syntax, such as `write_file` *`design_list`*.
`Courier bold`	Indicates user input—text you type verbatim—in examples, such as `prompt>` **`write_file top`**
[ ]	Denotes optional arguments in syntax, such as `write_file [-format` *`fmt`*`]`
...	Indicates that arguments can be repeated as many times as needed, such as *`pin1 pin2 ... pinN`*
|	Indicates a choice among alternatives, such as `low | medium | high`
Ctrl+C	Indicates a keyboard combination, such as holding down the Ctrl key and pressing C.
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy.

Feedback

Customer Support

Customer support is available through SolvNet online customer support and through contacting the Synopsys Technical Support Center.

Accessing SolvNet

The SolvNet site includes a knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. The SolvNet site also gives you access to a wide range of Synopsys online services including software downloads, documentation, and technical support.

To access the SolvNet site, go to the following address:

https://solvnet.synopsys.com

If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to sign up for an account.

If you need help using the SolvNet site, click HELP in the top-right menu bar.

Contacting the Synopsys Technical Support Center

If you have problems, questions, or suggestions, you can contact the Synopsys Technical Support Center in the following ways:

- Open a support case to your local support center online by signing in to the SolvNet site at https://solvnet.synopsys.com, clicking Support, and then clicking “Open A Support Case.”
- Send an e-mail message to your local support center.
  - E-mail support_center@synopsys.com from within North America.
  - Find other local support center e-mail addresses at

    https://www.synopsys.com/support/global-support-centers.html
- Telephone your local support center.
  - Call (800) 245-8005 from within North America.
  - Find other local support center telephone numbers at

    https://www.synopsys.com/support/global-support-centers.html

1

Low-Power Design Strategies

Power consumption is becoming an increasingly important aspect of circuit design. This chapter introduces power reduction strategies in the following sections:

- Increasing Challenges of Power
- Dynamic and Static Power
- Power Reduction Methods
- IEEE 1801 Standard (UPF)

Increasing Challenges of Power

Device densities and clock frequencies have increased dramatically in CMOS devices, thereby increasing power. At the same time, supply voltages and transistor threshold voltages have been lowered, causing leakage current to become significant.

High power consumption can result in excessively high temperatures during operation, reducing reliability because of electromigration and other heat-related failure mechanisms. High power consumption also reduces battery life in portable devices.

Energy is used to power millions of computers, servers, and other devices, both to run the devices and to cool the machines and buildings in which they are used. Even a small reduction in power consumption can result in large aggregate cost savings to users and significant benefits to the environment.

Dynamic and Static Power

Designers consider two types of power consumption, dynamic and static. Dynamic power is consumed during the switching of transistors, so it depends on the clock frequency and switching activity. Static power is the transistor leakage current that flows whenever power is applied to the device, so it is not related to switching activity.

The most advanced synthesis and physical implementation tools consider total power – the sum of dynamic and static leakage power – as well as physical placement optimization. Total power optimization shortens critical nets and cell sizing to optimize power within the allowed timing margins.

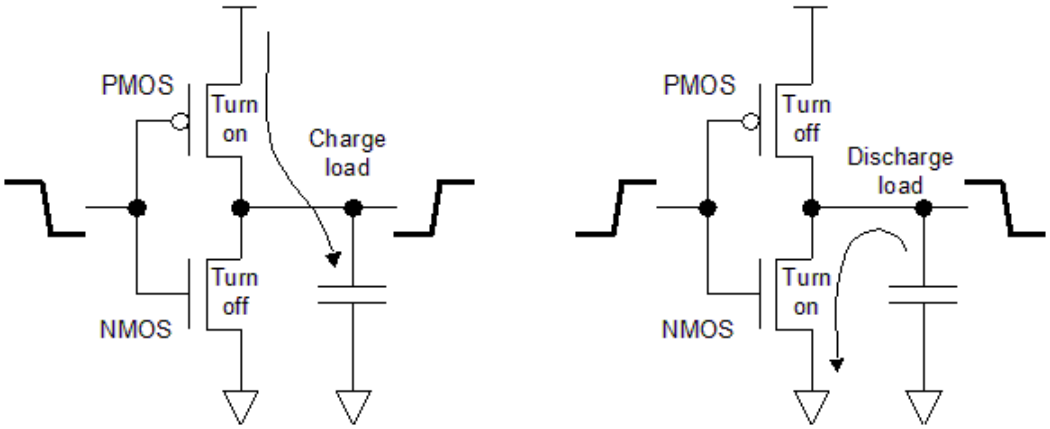
Feedback

Dynamic Power

Dynamic power is the energy used during logic transitions on nets, consisting of switching power and internal power. Switching power results from the charging and discharging of the external capacitive load on the cell output. Internal power results from the short-circuit (crowbar) current that flows through the PMOS-NMOS stack during a transition.

Switching power is illustrated in Figure 1. A transition from 0 to 1 on the inverter output charges the capacitive load through the PMOS transistor. A transition from 1 to 0 discharges the same capacitive load through the NMOS transistor.

Figure 1 Switching Power

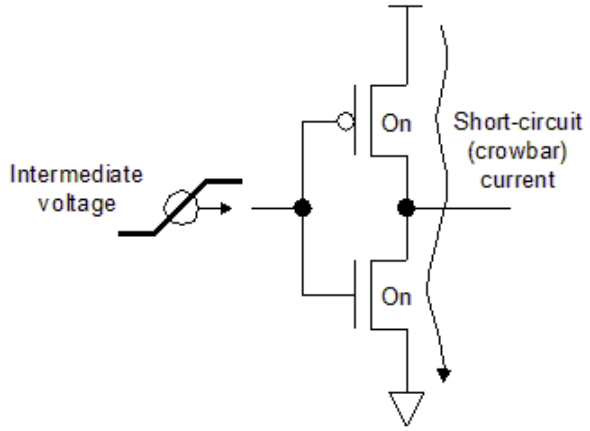


The energy used in each transition depends on the supply voltage and the capacitive load. Also, because the current flows only during logic transitions, the long-term dynamic power depends on the clock frequency and switching activity.

Internal power occurs when the input is at an intermediate voltage level, when both the PMOS and NMOS transistors are conducting. This condition results in a nearly short-circuit conductive path from VSS to ground, as illustrated in Figure 2. A relatively large current, called the crowbar current, flows through the transistors briefly. Lower threshold voltages and slower transitions result in more internal power.

Feedback

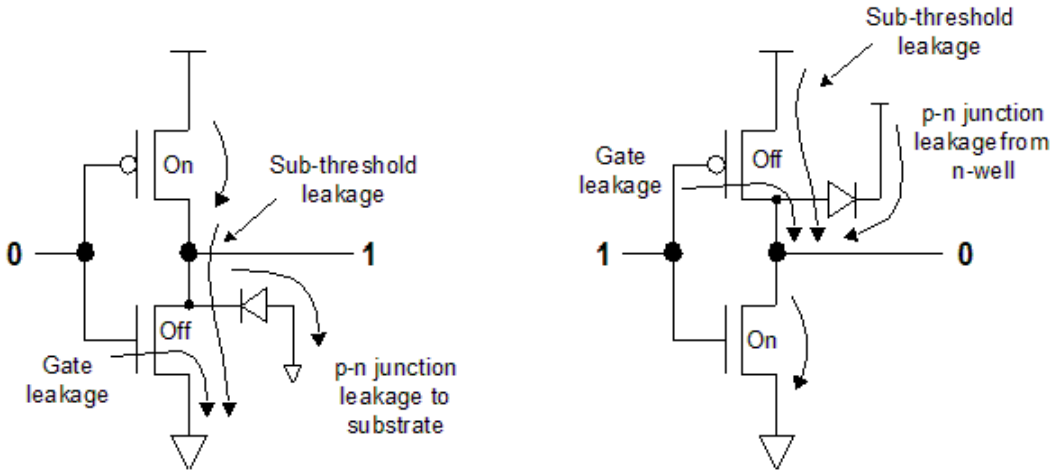
Figure 2 Internal Power



Static (Leakage) Power

With shrinking device geometries and reduced threshold voltages, leakage power is becoming more significant. Sub-threshold leakage, reverse-bias p-n junction diode leakage, and gate leakage all contribute to total leakage, as shown in Figure 3.

Figure 3 Static Leakage Currents



Sub-threshold leakage is the small source-to-drain current that flows even when the transistor is held in the “off” state. With lower power supply voltages and lower threshold voltages, “off” gate voltages are close to “on” threshold voltages. Sub-threshold leakage current increases exponentially as the gate voltage approaches the threshold voltage.

Leakage at reverse-biased p-n junctions (diode leakage) occurs from the n-type drain of the NMOS transistor to the grounded p-type substrate, and from the n-well (held at VDD) to the p-type drain of the PMOS transistor. This leakage is relatively small.

Gate leakage is the result of using an extremely thin insulating layer between the gate and channel of the MOS transistor. Quantum-effect tunneling of electrons through the gate oxide can occur, resulting in leakage from the gate to the source or drain.

Leakage currents occur whenever power is applied to the transistor, irrespective of the clock frequency or switching activity, but they can be reduced or eliminated by lowering the supply voltage or by switching off the power supply.

Power Reduction Methods

There are several different RTL and gate-level design strategies for reducing power. Some methods, such as clock gating, have been used widely and successfully for many years. Others, such as dynamic voltage and frequency scaling, have not been used much due to the difficulty of implementing them. As power becomes increasingly important, more methods are being exploited.

Supply Voltage Reduction

The most basic way to reduce power is to reduce the supply voltage. Power is proportional to the square of the supply voltage, for both dynamic and static power. Successive generations of CMOS technologies have used lower and lower supply voltages.

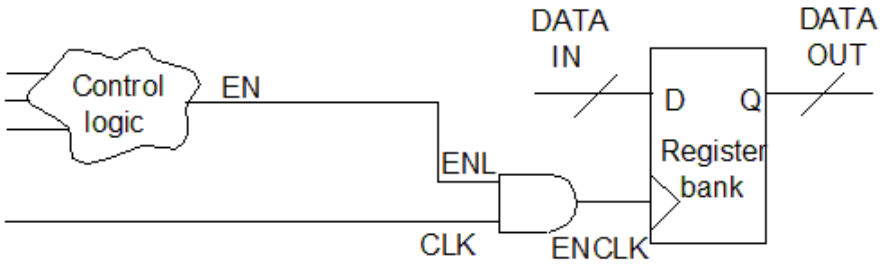
Each lowering of the supply voltage reduces per-gate power consumption, but also lowers the switching speed. In addition, the transistor threshold voltage must be lowered, causing more problems with noise immunity, crowbar currents, and sub-threshold leakage.

Clock Gating

Clock gating is a dynamic power reduction method that stops the clock signals for selected register banks during periods of inactivity. One simple implementation of clock gating is shown in Figure 4.

Feedback

Figure 4 Simple Clock Gating

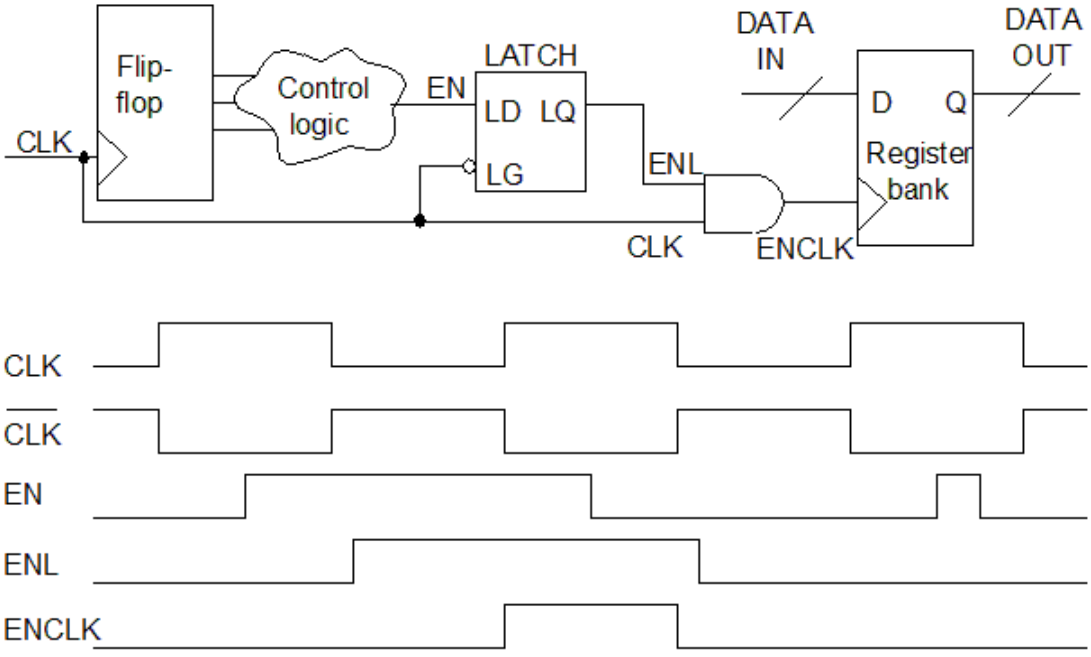


Clock gating is useful for registers that need to maintain the same logic values over many clock cycles. The main challenges are finding the best places to use it and creating the logic to shut off and turn on the clock at the proper times.

Synthesis tools such as Power Compiler can detect low-throughput datapaths where clock gating can give the greatest benefit and automatically insert clock-gating cells at the appropriate locations. Clock gating is relatively simple to implement because no additional power supplies or power infrastructure changes are needed.

Figure 5 shows an example of sequential, latch-based clock-gating cell and the related waveforms.

Figure 5 Latch-Based Clock Gating



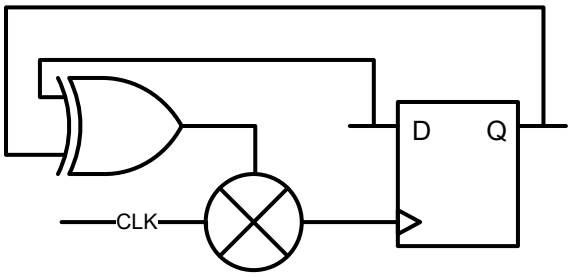
Feedback

The clock input to the register bank, ENCLK, is gated on or off by the AND gate. ENL is the enabling signal that controls the gating; it is derived from the EN signal of the clock gating control logic. The register bank is triggered by the rising edge of the ENCLK signal. Clock gating eliminates the need to reload the same value in the register through multiple clock cycles, thereby saving power.

Clock gating reduces clock network power dissipation, relaxes datapath timing, and reduces routing congestion by eliminating feedback multiplexer loops. For designs that have large register banks, clock gating can save power and area by reducing the number of gates in the design.

Another clock gating technique is self-gating, shown in Figure 6. In this example, an XOR gate compares the data stored in the register with the data arriving at the data pin of the register. The XOR output controls the enable condition for gating. If the data is unchanged, the unnecessary clock cycles are gated by the output of the XOR gate.

Figure 6 XOR Self-Gating Cell



Self-gating is useful when the enable condition cannot be inferred from the existing logic. It can be applied across multiple registers to create an combined enable condition for the register bank.

Multiple-Vt Library Cells

Some CMOS technologies have multiple cells operating at different threshold voltages (Vt values) for the same logic function. A low-Vt cell has better speed but higher sub-threshold leakage current.

The synthesis tool can choose the appropriate type of cell to use based on the tradeoff between speed and power. It uses low-Vt cells in the timing-critical paths for speed and high-Vt cells everywhere else for lower leakage power.

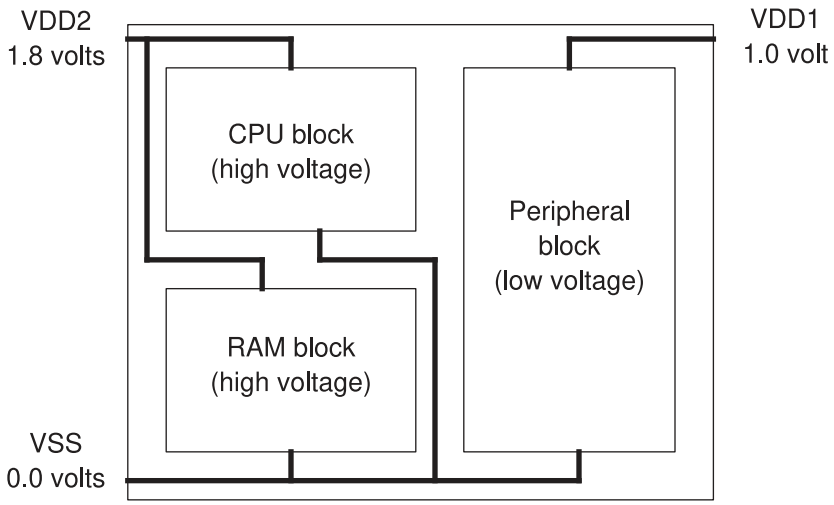
Multivoltage Design

Different parts of a chip might have different speed requirements. For example, the CPU and RAM blocks might need to be faster than a peripheral block. To get maximum speed where needed and also minimize power, the CPU and RAM can operate with a higher

Feedback

supply voltage while the peripheral block operates with a lower voltage, as shown in Figure 7.

Figure 7 Multivoltage Chip Design

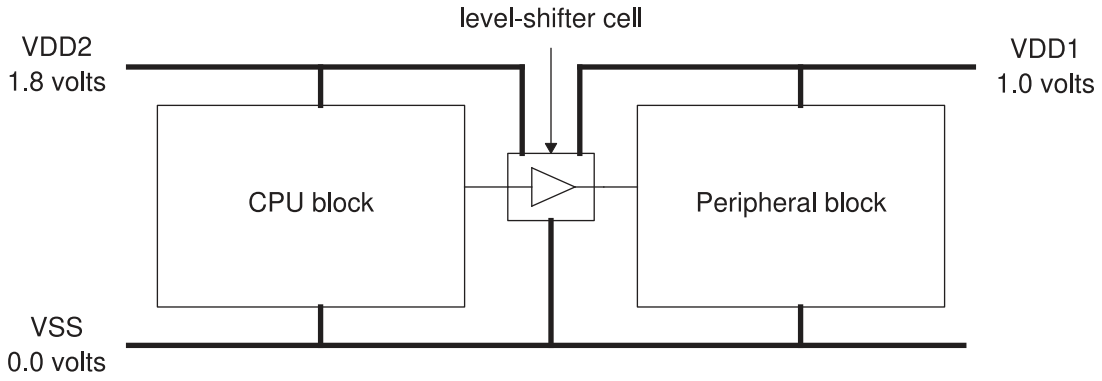


Providing multiple supply voltages on a single chip introduces some complexities and costs. Additional device pins must be available to supply the chip voltages, and the power grid must distribute each of the voltage supplies separately to the appropriate blocks.

Where a logic signal leaves one power domain and enters another, if the voltages are significantly different, a level-shifter cell is necessary to generate a signal with the proper voltage swing. In the example shown in Figure 8, a level shifter converts a signal with 1.8-volt swing to a signal with a 1.0-volt swing. A level-shifter cell itself requires two power supplies that match the input and output supply voltages.

Feedback

Figure 8 Level Shifter



Power Switching

Power switching is a power-saving technique that shuts down parts of the device during periods of inactivity. In a mobile phone chip, the block that performs voice processing can be shut down when the phone is in standby mode. The voice processing block must “wake up” from its powered-down state when needed.

Power switching has the potential to reduce overall power consumption substantially because it lowers leakage power as well as switching power. It also introduces some additional challenges:

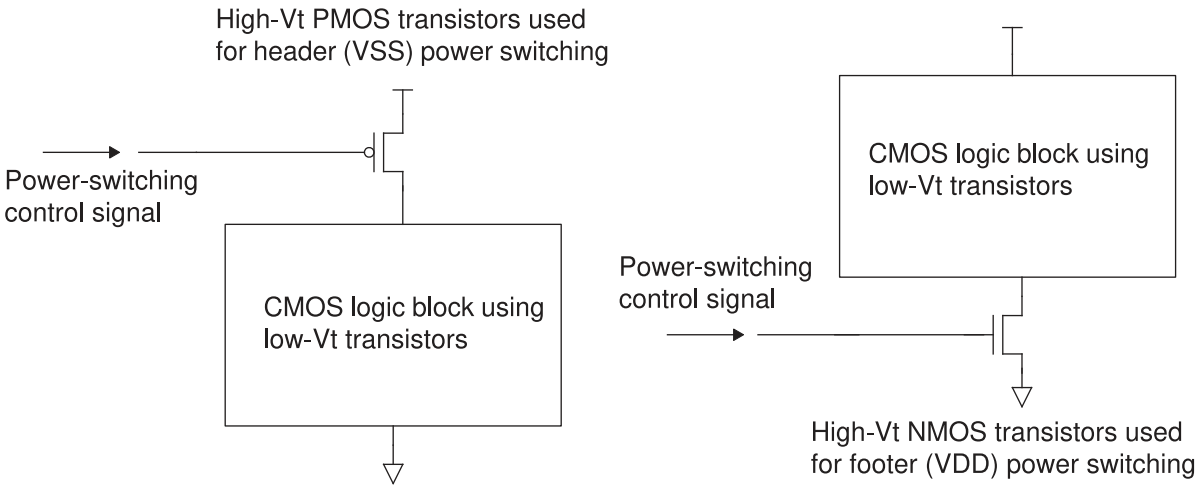
- Power controller – A logic block that determines when to power down and power up a specific block. There is some time and power cost for powering down and powering up a block, so the controller must determine the appropriate power-down times.
- Power switching network – A large number of high-Vt transistors with source-to-drain connections between the always-on power supply rail and the power pins of the cells. The power switches are distributed physically around or within the block. The network, when switched on, connects the power to the logic gates in the block.
- Isolation cells – Cells inserted in the design where signals leave a powered-down block and enter a block that is powered up. An isolation cell provides a known, constant logic value to an always-on block when the power-down block has no power, thereby preventing unknown or intermediate values that could cause crowbar currents.
- Retention registers – Registers that retain data during power-down by saving the data into a shadow register (also known as the bubble register). Upon power-up, the device restores the data from the shadow register to the main register.

Feedback

Power Switch Implementation

Power switches are implemented with high-Vt transistors because they minimize leakage and their speed is not critical. PMOS header switches can be placed between VDD and the block power supply pins, or NMOS footer switches can be placed between VSS and the block ground pins, as shown in Figure 9. The number, drive strength, and placement of switches should be chosen to give in an acceptable voltage drop during peak power usage.

Figure 9 Power-Switching Network Transistors



The switching strategy shown in Figure 9 is called a coarse-grain strategy because the power switching is applied to the whole block. Multiple transistors in parallel drive a common supply net for the block. In a fine-grain strategy, each library cell has its own power switch, allowing fine-grain control over which cells are powered down. The fine-grain approach has better potential for power savings, but requires significantly more area.

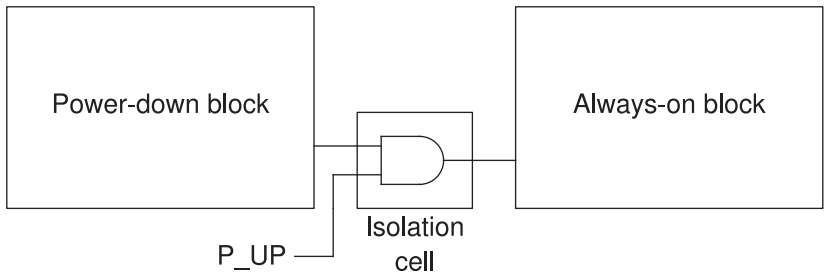
Power switching can be combined with multivoltage operation. Different blocks can be designed to operate at different voltages and also to be separately powered down when they are not needed. In that case, the interface cells between different blocks must perform both level shifting and isolation functions, depending on whether the two blocks are operating at different voltages or one is shut down. A cell that performs both functions is called an enable level shifter. This cell must have two separate power supplies, just like any other level shifter.

Feedback

Isolation Implementation

One simple implementation of an isolation cell is shown in Figure 10. When the block on the left is powered up, the signal P_UP is high and the output signal passes through the isolation cell unchanged (except for a gate delay). When the block on the left is powered down, P_UP is low, holding the signal constant at logic 0 going into the always-on block. Other types of isolation cells can hold a logic 1 rather than 0, or can hold the signal value latched at the time of the power-down event. Isolation cells must themselves have power during block power-down periods.

Figure 10 Isolation Cell



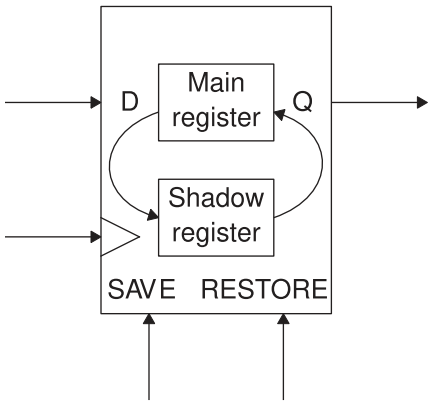
Retention Cell Implementation

When a block is powered down and then powered back up, it is often desirable for the block to be restored to the state it was in before the power-down event. There are several strategies for doing this. For example, the block register contents can be copied to RAM outside of the block before power-down, and then copied back after power-up.

Another strategy is to use retention registers in the power-down block. One type of retention register implementation is shown in Figure 11. The SAVE signal saves the register data into the shadow register before power-down and the RESTORE signal restores the data after power-up. Instead of using separate, edge-sensitive SAVE and RESTORE signals, a retention register could use a single level-sensitive control signal.

Feedback

Figure 11 Retention Register



A retention register occupies a larger area than an ordinary register, and it requires an always-on power supply connection for the shadow register in addition to the power-down supply used by the rest of the device. However, restoring the data to the registers after power-up is fast and simple compared with other strategies.

Dynamic Voltage and Frequency Scaling

The principle of multivoltage operation can be extended to allow the voltage to be changed during operation of the chip to match the current workload. For example, a math processor chip might operate at a lower voltage and clock frequency during simple spreadsheet computations, and then at a higher voltage and clock frequency during 3-D image rendering. The changing of supply voltage and operating frequency to meet workload requirements is called dynamic voltage and frequency scaling.

The chip and voltage supply can be designed to use a number of established levels, or even a continuous range. Dynamic voltage scaling requires a multilevel power supply and a logic block to determine the best voltage level to use for a given task. Design, implementation, verification, and testing of the device can be challenging because of the ranges and combinations of voltage levels and operating frequencies that must be analyzed.

Power-Optimized IP Components

IP collections such as Synopsys DesignWare® minPower components are available to automatically implement datapath architectures that minimize high-activity datapaths and suppress glitches. This reduces both dynamic and leakage power for complex combinational logic such as that used in multipliers and digital signal processors.

Based on the actual switching activity, transition probabilities, available standard cells, and analysis of possible configurations, the synthesis tool automatically configures the IP component architectures to implement the optimal structure with the lowest power consumption. The IP components include blocks that incorporate low-power design techniques such as enhanced clock gating, built-in datapath gating, and data-tracking pipeline management technology.

Multibit Register Synthesis and Implementation

Synthesis and physical implementation tools group individual register bits into multibit registers, allowing a single clock input to drive multiple register bits. This reduces the need for clock tree resources such as buffers and wire, thereby reducing both power and area. The multibit cell itself is more efficient by sharing logic, power supply connections, and transistor wells for the individual bits.

To get the best possible power savings during multibit synthesis, provide accurate switching activity data in a SAIF file. This allows the synthesis tool to map bits into multibit registers according to switching activity so that total power is minimized.

IEEE 1801 Standard (UPF)

The IEEE 1801 Standard for Design and Verification of Low Power Integrated Circuits, also known as the Unified Power Format (UPF), consists of a set of commands used to specify the power intent for multivoltage electronic systems. Using UPF commands, you can specify the supply network, power switches, isolation, retention, and other aspects of power management for a chip design. A single set of low-power design specification commands is used throughout the design, analysis, verification, and implementation flow.

Synopsys tools support the UPF infrastructure and commands. Usage of the UPF commands throughout the multivoltage flow is the main focus of this book. The term “multivoltage” is intended to include multisupply designs (those with shut-down power domains), even when the multiple domains operate at the same voltage.

2

Library Requirements

In order to use power-saving strategies such as clock gating, multivoltage, multiple-Vt library cells, or power switching, the library must contain logic cells that support these strategies. Some of the types of cells that support low-voltage design are described in the following sections:

Liberty PG Pin Syntax

In earlier generations of CMOS technologies, all devices on a chip were connected to a single, chip-wide power supply at all times. Logic libraries did not contain information about power supply connections of cells because all cells shared the same types of connections to VDD and VSS.

However, with the increasing use of multiple power supplies on a chip, it has become necessary to specify which power supplies can be connected to specific power pins of each cell. For some types of cells such as level shifters, specifying different power supplies for different power pins of the same cell has become necessary.

To accommodate this type of information, the Liberty library syntax was expanded to support the specification of power rail connections to the power supply pins of cells. This power and ground (PG) pin information allows synthesis, implementation, and verification

Feedback

tools to optimize the design for power, to properly connect the cells in layout, and to analyze the design behavior where multiple supply voltages are being used.

For specific information about the PG pin syntax and the modeling of power supply pin connections, see the chapter called “Advanced Low-Power Modeling” in the *Library Compiler User Guide*.

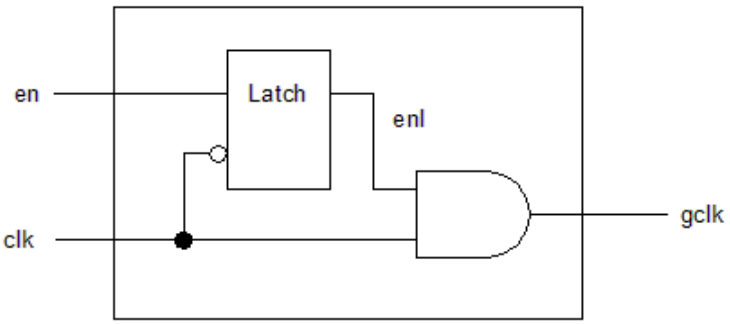
For an older library that does not have PG pins, you can quickly add PG pins with the `add_pg_pin_to_lib` or `add_pg_pin_to_db` command in Design Compiler or IC Compiler, thereby making the library compatible with the UPF power specification. Alternatively, you can use the `update_lib_model` command to update the library using FRAM models or the `update_lib_voltage_model`, `update_lib_pg_pin_model`, and `update_lib_pin_model` commands to provide the PG pin information. For details, see Converting Libraries to PG Pin Library Format.

Clock-Gating Cells

Synthesis tools such as Power Compiler can determine where clock gating can be used to provide the greatest power-saving benefit, and can automatically insert clock-gating circuits into the design to implement the clock-gating functions.

Inserting clock-gating circuitry into an existing clock network can introduce skew that adversely affects timing. To have the synthesis tool account for such effects during synthesis, you can have the tool use predefined integrated clock-gating cells, which can be provided as logic cells in the library. An integrated clock-gating cell integrates the various combinational and sequential elements of a clock gate into a single library cell. Figure 12 shows one possible implementation of an integrated clock-gating cell.

Figure 12 Integrated Clock-Gating Cell Example



A clock-gating cell can incorporate any kind of logic such as multiple enable inputs, test clock input, global scan input, asynchronous reset latch, active-low enabling logic, or inverted gated clock output. Power Compiler is free to optimize the enabling logic

Feedback

surrounding a clock-gating cell by absorbing the surrounding logic into the logic functions available inside the cell.

For more information about creating and using integrated clock-gating cells, see the chapter called “Modeling Power and Electromigration” in the *Library Compiler User Guide*; and the chapter called “Clock Gating” in the *Power Compiler User Guide*.

Multiple Threshold-Voltage Cells

Multiple threshold voltage libraries support two or more different threshold-voltage groups for each logic gate. The threshold voltage determines the delay and leakage characteristics of the logic cell. Cells with lower threshold voltages can switch more quickly but have higher leakage. Cells with higher threshold value have less leakage but a longer switching delay.

Liberty library syntax supports cells of different threshold voltages in the same library using the `default_threshold_voltage_group` attribute. This attribute defined at the library level specifies the name of the category to which a cell belongs, based on the voltage characteristics of the cell. This attribute has the following syntax:

```
default_threshold_voltage_group : "group_name";
```

The following example shows the usage of the `default_threshold_voltage_group` attribute.

```
default_threshold_voltage_group : "high_vt_cell";
```

Liberty library syntax also supports an optional cell-level `threshold_voltage_group` attribute that you can use to associate a cell with one or more threshold-voltage categories as shown in the following example:

```
cell (AND1_H) {
   ...
   threshold_voltage_group "high_vt_cell";
   ...
}
cell (AND1_L) {
   ...
   threshold_voltage_group "low_vt_cell";
   ...
}
```

Feedback

Defining Multiple Threshold-Voltage Cells Using Attributes

If your target library does not define multiple threshold-voltage cells and the associated attributes, you can use the `set_attribute` command in the tool to define multiple threshold-voltage cells.

The following Design Compiler script example shows how you set the library-level `default_threshold_voltage_group` attribute to the high_vt_library and low_vt_library library files, which contain cells of the high and low threshold-voltage groups respectively in separate library files.

```
set HVT_lib "high_vt_library"
set LVT_lib "low_vt_library"
set_attribute [get_libs $HVT_lib] -type string \
   default_threshold_voltage_group HVT
set_attribute [get_libs $LVT_lib] -type string \
   default_threshold_voltage_group LVT
```

If you have multiple threshold-voltage cells in the same library file, you can set the `default_threshold_voltage_group` and `threshold_voltage_group` attributes as shown in the following example:

```
set all_vt_lib "multiple_vt_lib"
set_attribute [get_libs $all_vt_lib] -type string \
   default_threshold_voltage_group HVt
set_attribute [get_lib_cells ${all_vt_lib}/FAST*] -type string \
   threshold_voltage_group LVt
set_attribute [get_lib_cells ${all_vt_lib}/MEM*] -type string \
   threshold_voltage_group MD
```

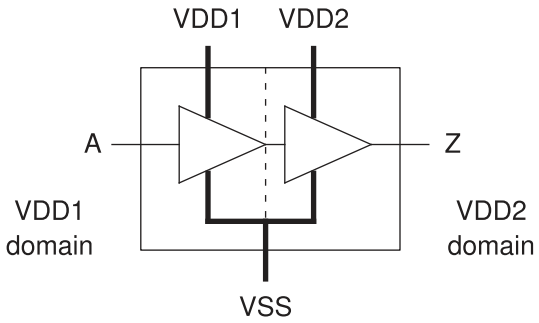
To set the multithreshold voltage constraint in the Design Compiler or IC Compiler tool, use the `set_multi_vth_constraint` command, or in the IC Compiler II or Fusion Compiler tool, use the `set_max_lvth_percentage` command. For details, see the man page for the command.

Level-Shifter Cells

In a multivoltage design, a level shifter is required where each signal crosses from one power domain to another. The level shifter operates as a buffer with one supply voltage at the input and a different supply voltage at the output. Thus, a level shifter converts a logic signal from one voltage swing to another, with a goal of having the smallest possible delay from input to output. See Figure 13.

Feedback

Figure 13 Level Shifter



The library description of a level-shifter cell must have information about the type of conversion performed (high-to-low, low-to-high, or both), the supported voltage levels, and the identities of the respective power pins that must be connected to each power supply.

Synopsys synthesis and implementation tools can identify nets in the design that require adjustment between the driver and receiver, find appropriate level-shifter cells in the available libraries, insert the level shifters into the netlist, place the level shifters, and route the required logic connections and respective power supplies.

The following example shows the form of the Liberty syntax for a buffer-type low-to-high level shifter.

```
cell(Buffer_Type_LH_Level_shifter) {
  is_level_shifter : true;
  level_shifter_type : LH ;
  pg_pin(VDD1) {
    voltage_name : VDD1;
    pg_type : primary_power;
    std_cell_main_rail : true;
  }
  pg_pin(VDD2) {
    voltage_name : VDD2;
    pg_type : primary_power;
  }
  pg_pin(VSS) {
    voltage_name : VSS;
    pg_type : primary_ground;
  }
  ...
  pin(A) {
    direction : input;
    related_power_pin : VDD1;
    related_ground_pin : VSS;
    input_voltage_range ( 0.7 , 0.9);
  }
  pin(Z) {
```

Feedback

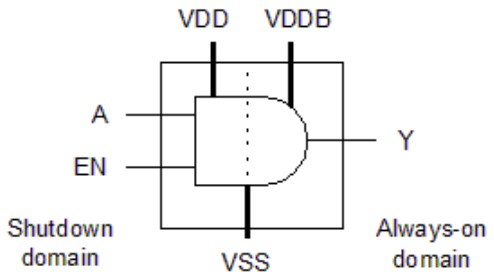
```
    direction : output;
    related_power_pin : VDD2;
    related_ground_pin : VSS;
    function : "A";
    power_down_function : "!VDD1 + !VDD2 + VSS";
    output_voltage_range (1.1 , 1.3);
    ...
  }
 ...
}
```

For more information about creating and using level-shifter cells, see the chapter called “Advanced Low-Power Modeling” in the *Library Compiler User Guide*; and the *Power Compiler User Guide*.

Isolation Cells

In a design with power switching, an isolation cell is required where each logic signal crosses from a power domain that can be powered down to a domain that is not powered down. The cell operates as a buffer when the input and output sides of the cell are both powered up, but provides a constant output signal during times that the input side is powered down. An enable input controls the operating mode of the cell. See Figure 14.

Figure 14 Isolation Cell



The following example shows the form of the Liberty syntax for a typical isolation cell.

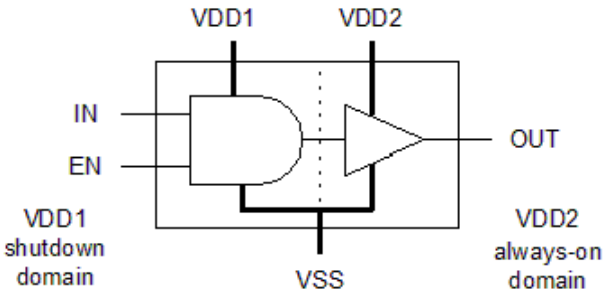
```
cell(Isolation_Cell) {
is_isolation_cell : true;
dont_touch : true;
dont_use : true;
pg_pin(VDD) {
  voltage_name : VDD;
  pg_type : primary_power;
}
pg_pin(VSS) {
  voltage_name : VSS;
```

Feedback

```
    pg_type : primary_ground;
  }
  ...
  pin(A) {
    direction : input;
    related_power_pin : VDD;
    related_ground_pin : VSS;
    isolation_cell_data_pin : true;
  }
  pin(EN) {
    direction : input;
    related_power_pin : VDD;
    related_ground_pin : VSS;
    isolation_cell_enable_pin : true;
  }
  pin(Y) {
    direction : output;
    related_power_pin : VDD;
    related_ground_pin : VSS;
    function : "A * EN";
    power_down_function : "!VDD + VSS";
    timing() {
      related_pin : "A EN";
      cell_rise(template) {
        ...
      }
    }
    ...
  }
  ...
  }
```

A cell that can perform both level-shifting and isolation functions is called an enable level-shifter cell. This type of cell is used where a signal crosses from one power domain to another, where the two voltage levels are different and the first domain can be powered down. See Figure 15.

Figure 15 Enable Level Shifter



Feedback

The following example shows the form of the Liberty syntax for a typical enable level shifter.

```
cell(Enable_Level_Shifter) {
  is_level_shifter : true;
  level_shifter_type : LH ;
  input_voltage_range(0.7,1.4);
  output_voltage_range(0.7,1.4);

pg_pin(P1) {
  voltage_name : VDD1;
  pg_type : primary_power;
  std_cell_main_rail : true;
}
pg_pin(P2) {
  voltage_name : VDD2;
  pg_type : primary_power;
}
...
pin(A) {
  direction : input;
  related_power_pin : P1;
  related_ground_pin : G1;
  level_shifter_data_pin:true;
}
pin(EN) {
  direction : input;
  related_power_pin : P1;
  related_ground_pin : G1;
  level_shifter_enable_pin:true;
}
...
}
```

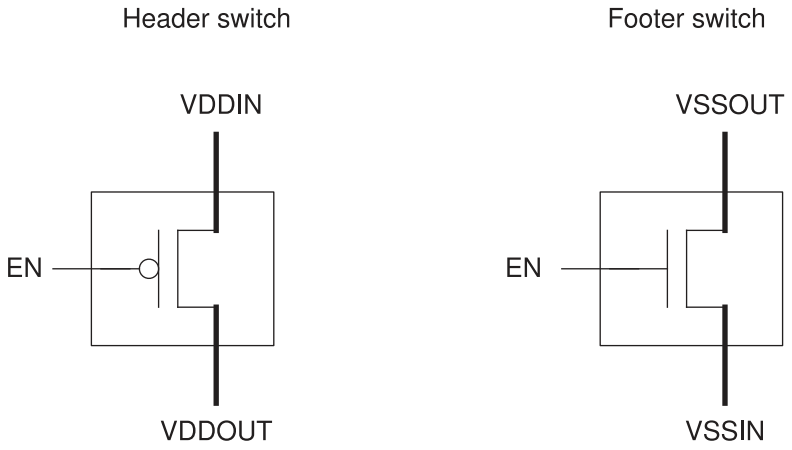
For more information about creating and using isolation cells and enable level-shifter cells, see the chapter called “Advanced Low-Power Modeling” in the *Library Compiler User Guide*; and the *Power Compiler User Guide*.

Power-Switch Cells

In a design with power switching, either header or footer type power switch cells are required to supply power for cells that can be powered down. A header type power switch connects the power rail to the power supply pins of the cells in the power-down domain. A footer type power switch connects the ground rail to the ground supply pins of the cells in the power-down domain. An input logic signal to the power switch controls the connection or disconnection state of the switch. See Figure 16.

Feedback

Figure 16 Power-Switch Cells



The library description of a power-switch cell specifies the input signal that controls power switching, the pin or pins connected to the real power rail, and the pin or pins that provide the virtual (switchable) power. The power-switch cell can optionally have an output “acknowledge” signal indicating the current status of the switch.

The following example shows the form of the Liberty syntax for a typical power-switch cell.

```
cell ( Simple_CG_Switch ) {
  ...
  switch_cell_type : coarse_grain;

  pg_pin ( VDD ) {
    pg_type : primary_power;
    direction : input;
    voltage_name : VDD;
  }
  pg_pin ( VVDD ) {
    pg_type : internal_power;
    voltage_name : VVDD;
    direction : output ;
    switch_function : "SLEEP" ;
    pg_function : "VDD" ;
  }
  pg_pin ( VSS ) {
    pg_type : primary_ground;
    direction : input;
    voltage_name : VSS;
  }
  ...
  pin ( SLEEP ) {
    switch_pin : true;
    capacitance: 1.0;
```

Feedback

```
    related_power_pin : VDD;
    related_ground_pin : VSS;
  }
  ...
  }
```

For more information about creating power-switch cells, see the chapter called “Advanced Low-Power Modeling” in the *Library Compiler User Guide*. For information about using these cells in physical planning, see the *IC Compiler Design Planning User Guide*, *IC Compiler II Design Planning User Guide*, or *Fusion Compiler Design Planning User Guide*.

Fine-Grain Switch Cells

Synopsys synthesis and implementation tools support macro cells with fine-grain switches that have the following attribute settings in the PG pin definition in the library:

- The `direction` attribute is `internal`.
- The `pg_type` attribute is either `internal_power` or `internal_ground`.
- The `pg_function` attribute is defined.
- The `switch_function` attribute is defined.
- The `switch_cell_type` attribute of the macro is `fine_grain`.
- The `switch_pin` attribute is set to `true` for the control port.

In the tool that uses the macro cell, use the `connect_supply_net` command to connect to the internal PG pins of these macro cells. Supply nets connected only to the internal PG pins of these macro cells cannot be used for level-shifter insertion and always-on synthesis, unless the following conditions are true:

- The supply net is the primary supply of the power domain.
- The supply net is specified by the isolation strategy of the power domain.
- The supply net is specified by the retention strategy of the power domain.
- The supply net is defined or reused as a domain-dependent supply net of the power domain.
- The supply net is defined with the `extra_supplies_#` keyword.

To specify the operating voltage of a fine-grain switch cell, use the `set_voltage` command.

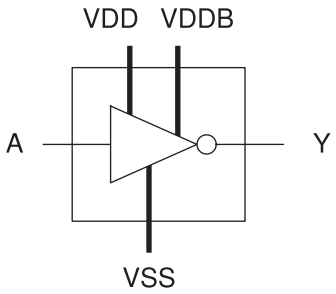
For more information about defining fine-grain switch cells, see the chapter called “Advanced Low-Power Modeling” in the *Library Compiler User Guide*.

Feedback

Always-On Logic Cells

When dealing with shutdown domains, there can be some situations in which certain cells in the shutdown portion need to continuously stay active, such as for implementing retention registers, isolation cells, retention control paths, and isolation enable paths. For example, if a save signal or restore signal passing through a shutdown voltage area needs buffering, an always-on buffer cell must be used. This type of logic is called always-on logic, which is built with always-on library cells. Compared to an ordinary cell, a functionally equivalent always-on cell has a backup power supply that operates continuously, even during the shutdown mode. See Figure 17.

Figure 17 Always-On Logic Cell



The following example shows the form of the Liberty syntax for an always-on buffer.

```
cell(buffer_type_AO) {
  always_on : true;
  pg_pin(VDD) {
    voltage_name : VDD;
    pg_type : primary_power;
  }
  pg_pin(VDDB) {
    voltage_name : VDDB;
    pg_type : backup_power;
  }
  pg_pin(VSS) {
    voltage_name : VSS;
    pg_type : primary_ground;
  }
  ...
  pin (A) {
    related_power_pin : VDDB;
    related_ground_pin : VSS;
  }
  pin (Y) {
    function : "A";
    related_power_pin : VDDB;
```

Feedback

```
  related_ground_pin : VSS;
  power_down_function : "!VDDB + VSS";
}
...
```

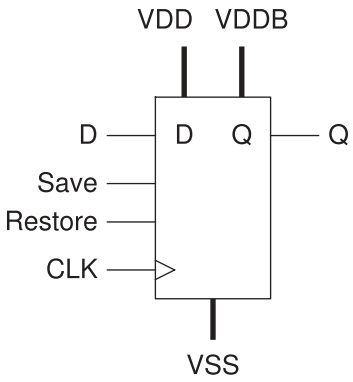
For more information about creating and using always-on logic cells, see the chapter called “Advanced Low-Power Modeling” in the *Library Compiler User Guide*; and the *Power Compiler User Guide*.

Retention Register Cells

In a design with power switching, there are several different ways to save register states before power-down and restore them upon power-up in the power-down domain. One method is to use retention registers, which are registers that can maintain their state during power-down by means of a low-leakage register network and an always-on power supply.

The library description of a retention register specifies the power pins and the input signals that control the saving and restoring of data. It also specifies which power pins are normal and can be powered down and which ones are the always-on pins used to maintain the data during power-down. See Figure 18.

Figure 18 Retention Register



The following example shows the form of the Liberty syntax for a typical retention register.

```
cell(RETENTION_DFF) {
retention_cell:"ret_dff";
  area : 1.0;
  ...
  pg_pin(VDDB) {
    voltage_name : VDDB;
    pg_type : backup_power;
```

Feedback

```
}
...
pin(RETN) {
  direction : input;
  capacitance : 1.0;
  nextstate_type : data ;
  related_power_pin :VDDB;
  related_ground_pin:VSSG;
  retention_pin (save_restore, "1" );
  }
pin(Q) {
  power_down_function:"!VDD+VSS";
  related_power_pin : VDD ;
  related_ground_pin : VSS;
  direction : output;
  ...
```

For more information about retention register cells, see the *Power Compiler User Guide*.

Macro Cells With Internally Generated Power

A macro cell can contain a voltage converter subcircuit that supplies power to other subcells within the macro cell at a voltage different from the supply voltage provided to the macro cell itself. The internally generated supply can be represented by an internal PG pin.

Synopsys synthesis and implementation tools support macro cells with internal PG pins. These macro cells have the following attribute settings in the PG pin definition in the library:

- The `direction` attribute is `internal`.
- The `pg_type` attribute is `internal_power`.
- The `pg_function` attribute is set to the name of the macro cell supply from which the internal voltage converter generates the internal voltage.
- The `switch_cell_type` attribute is not defined; the macro cell uses the generated supply voltage internally and does not make it available as an output.

For more information about macros with internal PG pins, see the chapter called “Advanced Low-Power Modeling” in the *Library Compiler User Guide*.

Converting Libraries to PG Pin Library Format

If the target library that you specify complies with the power and ground (PG) pin Liberty library syntax, the tool uses this information during synthesis. If your target library does not

Feedback

already contain PG pin information, you can convert it to PG pin library format by using the following methods:

- Using the FRAM View
- Using Tcl Commands
- Commands for Low-Power Library Specification

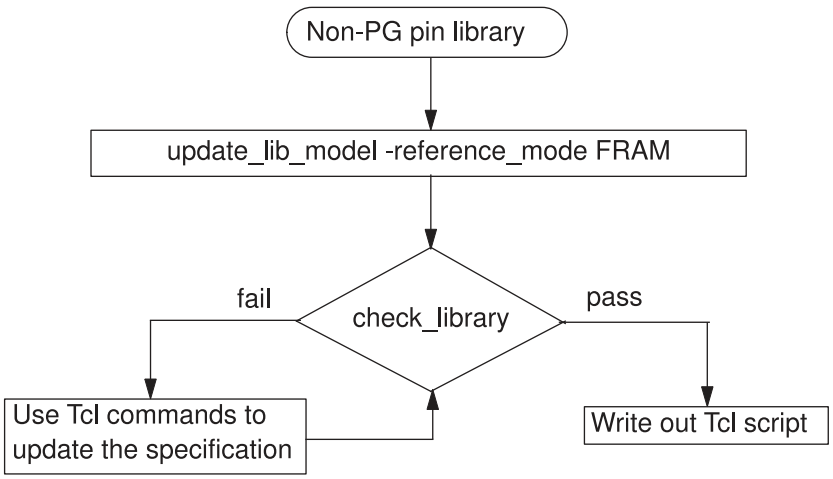
For more information, see SolvNet article 029641, “On-the-Fly Low-Power Library Specification”.

The NDM library preparation flow directly accepts logic libraries without PG pin data. The library manager resolves missing PG pins during library preparation by using information in the LEF file.

Using the FRAM View

In Design Compiler topographical mode or IC Compiler tool, you can use the FRAM view as the reference for converting your library to PG pin library format. You must set the `mw_reference_library` variable to the location of the Milkyway reference libraries. Use the `update_lib_model` command to convert your library to PG pin library format. The tool uses the PG pin definitions available in the FRAM view of the Milkyway library for the conversion. This is the default behavior. Figure 19 shows the steps involved in converting a non-PG pin library to a PG pin library.

Figure 19 Conversion of a Non-PG Pin Library to a PG Pin Library Using FRAM View



To verify that your new PG pin library is complete, use the `check_library` and `report_mv_library_cells` commands. If the PG pin library is not complete, run the

library specification Tcl commands to complete the library creation. For more information, see Commands for Low-Power Library Specification.

Using Tcl Commands

When your library files are not in PG pin library syntax, and you do not have FRAM views from the Milkyway library, you can use the following Tcl commands to specify the information required to derive the PG pin details, as shown in Figure 20.

- `update_lib_voltage_model` *`library_name`* `-voltage ...`

  This command sets the voltage map for the specified library.

- `update_lib_pg_pin_model` *`cell_name`* `-pg_pin_name` *`pin_name`* `...`

  This command sets the PG pin map for the specified library cell.

- `update_lib_pin_model` *`cell_name`* `-pins` *`pin_list`* `...`

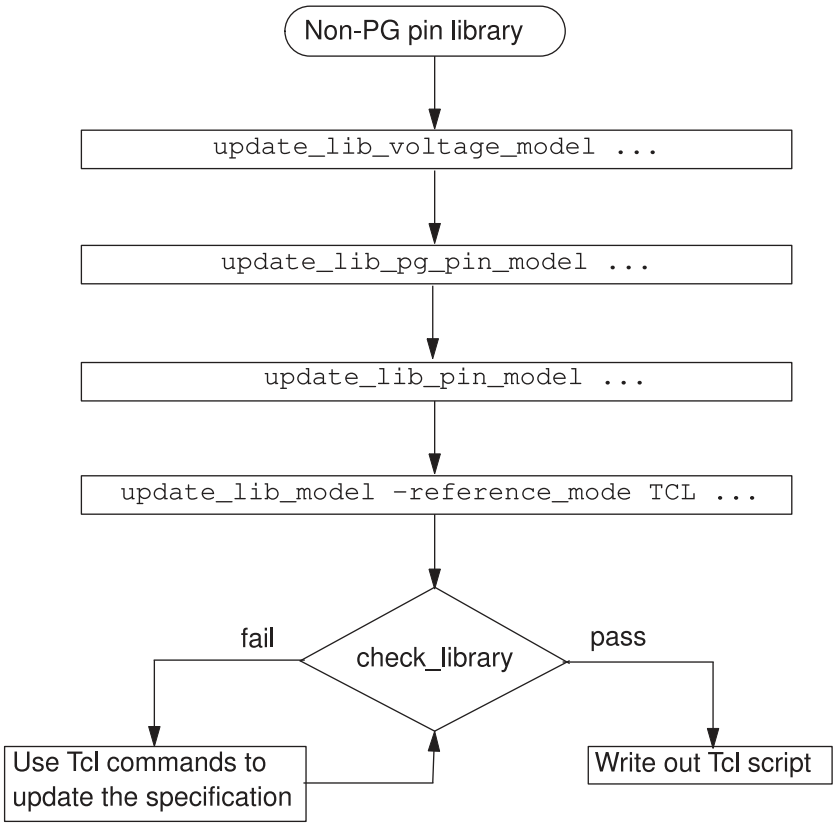
  This command sets the pin map for the specified library cell.

- `update_lib_model -reference_mode TCL` *`library_name`*

  This command updates the library to conform to PG pin library syntax.

Feedback

Figure 20 Conversion of Non-PG Pin Library to PG Pin Library Using Tcl Commands



These Tcl commands specify the library requirements that are used while converting the libraries to PG pin format.

Run the `update_lib_model -reference_mode TCL` command to convert your libraries to PG pin library format. To check if your newly created PG pin library is complete, run the `check_library` command. If your newly created PG pin library contains conflicts or is incomplete, run the library specification Tcl commands to complete the library specification.

Commands for Low-Power Library Specification

When you convert your library to PG pin format, if the newly created library file is complete, you can start using the library for the low-power implementation of your design. However, if your library contains power management cells, and the modeling is not

complete, you can use the following Tcl commands to complete your library specifications. These commands specify the library voltage and PG pin characteristics.

- `set_voltage_model ...`

  This command sets the voltage model on the specified library by updating the voltage map in the library.

- `set_pg_pin_model ...`

  This command defines the PG pins for the specified cell.

- `set_pin_model ...`

  This command defines the related power, ground, or bias pins of the specified pin of the library.

For more details, see the command man page and the Library Checking Chapter in the *Library Quality Assurance System User Guide*.

3

Power Intent Specification

The IEEE 1801 Standard for Design and Verification of Low Power Integrated Circuits, also known as the Unified Power Format (UPF), provides a consistent way to specify power implementation intent throughout the design process, including synthesis, physical implementation, and verification. This consistency makes it easier to perform synthesis, simulation, logical equivalence checking, and design verification in the presence of specific low-power features in a given design.

Specifying the power intent of a design in the Synopsys flow is described in the following sections:

IEEE 1801 Standard (UPF)

The IEEE 1801 Standard for Design and Verification of Low Power Integrated Circuits, also known as the Unified Power Format (UPF), consists of a set of Tcl-like commands used to specify the design intent for multivoltage electronic systems. Using UPF commands, you can specify the supply network, switches, isolation, retention, and

Feedback

other aspects of power management for a chip design. A single set of low-power design specification commands can be used throughout the design, analysis, verification, and implementation flow.

Synopsys tools are designed to follow the IEEE 1801 (UPF) standard approved by the IEEE Standards Association. This standard is supported by a large number of EDA companies, including Synopsys. Detailed information about the standard can be found in the IEEE 1801 specification itself, available as a download from the IEEE Standards Association at http://standards.ieee.org.

Synopsys tools support a large subset of the commands in the IEEE 1801 (UPF) standard. In addition, they support some UPF-like power intent commands that are not part of the standard.

Wildcard Pattern Matching in UPF Commands

The IEEE 1801 LRM states that wildcard pattern matching is only allowed in the `find_objects` and `upf_query` commands. However, Synopsys tools provide wildcard matching for many UPF commands to improve usability.

Wildcard pattern matching uses the following special operators:

- ? matches any single character, except the hierarchical separator "/"
- * matches any sequence of zero or more characters, except the hierarchical separator.

All UPF commands that perform pattern-based object lookup support the "?" and "*" wildcards, and the object lookups occur in the current scope. Table 1 shows some examples of wildcard matching.

Table 1 Examples of Wildcard Pattern Matching

Pattern	Matches
a	a
e*	e12, eac, ef
d?f	daf, d4f
a/b*/c*	a/baa/c1aa

Power Intent Concepts

The UPF language provides a way to specify the power requirements of a design, but without specifying explicitly how those requirements are implemented. The language specifies how to create a power supply network to each design element, the behavior

of supply nets with respect to each other, and how the logic functionality is extended to support dynamic power switching to design elements. It does not contain any placement or routing information. The UPF specification is separate from the RTL description of the design.

In the UPF language, a *power domain* is a group of elements in the design that share a common set of power supply needs. By default, all logic elements in a power domain use the same primary supply and primary ground. Other power supplies can be defined for a power domain as well. A power domain is typically implemented as a contiguous *voltage area* in the physical chip layout, although this is not a requirement of the language.

Each power domain has a *scope* and an *extent*. The *scope* is the level of logic hierarchy designated as the root of the domain. The *extent* is the set of logic elements that belong to the power domain and share the same power supply needs. The scope is the hierarchical level at which the domain is defined and is an ancestor of the elements belonging to the power domain, whereas the extent is the actual set of elements belonging to the power domain.

Each scope in the design has *supply nets* and *supply ports* at the defined hierarchical level of the scope. A *supply net* is a conductor that carries a supply voltage or ground throughout a given power domain. A supply net that spans more than one power domain is said to be “reused” in multiple domains. A *supply port* is a power supply connection point between two adjacent levels of the design hierarchy, between the parent and child blocks of the hierarchy. A supply net that crosses from one level of the design hierarchy to the next passes through a supply port.

A *supply set* is an abstract collection of supply nets, consisting of at least two supply functions, power and ground. A supply set is *domain-independent*, which means that the power and ground in the supply set are available to be used by any power domain defined within the scope where the supply set was created. However, each power domain can be *restricted* to limit its usage of supply sets within that power domain.

You can use supply sets to define power intent at the RTL level, so you can synthesize a design even before you know the names of the actual supply nets. A supply set is an abstraction of the supply nets and supply ports needed to power a design. Before such a design can be physically implemented (placed and routed), its supply sets must be *refined*, or associated with actual supply nets.

A *supply set handle* is an abstract supply set created for a power domain. By default, a power domain has supply set handles for the domain’s primary supply set, a default isolation supply set, and a default retention supply set. These supply set handles let you synthesize a design even before you create any supply sets, supply nets, and supply ports for the power domain. Before such a design can be physically implemented, its supply set handles must be *refined*, or associated with actual supply sets; and those supply sets must be refined so that they are associated with actual supply nets.

A *power switch* (or simply *switch*) is a device that turns on and turns off power for a supply net. A switch has an input supply net, an output supply net that can be switched on or off, and at least one input signal to control switching. The switch can optionally have multiple input control signals and one or more output acknowledge signals. A *power state table* lists the allowed combinations of voltage values and states of the power switches for all power domains in the design.

A *level shifter* must be present where a logic signal leaves one power domain and enters another at a substantially different supply voltage. The level shifter converts a signal from the voltage swing of the first domain to that of the second domain.

An *isolation* cell must be present where a logic signal leaves a switchable power domain and enters a different power domain. The isolation cell generates a known logic value during shutdown of the domain. If the voltage levels of the two domains are substantially different, the interface cell must be able to perform both level shifting (when the domain is powered up) and isolation (when the domain is powered down). A cell that can perform both functions is called an *enable level shifter*.

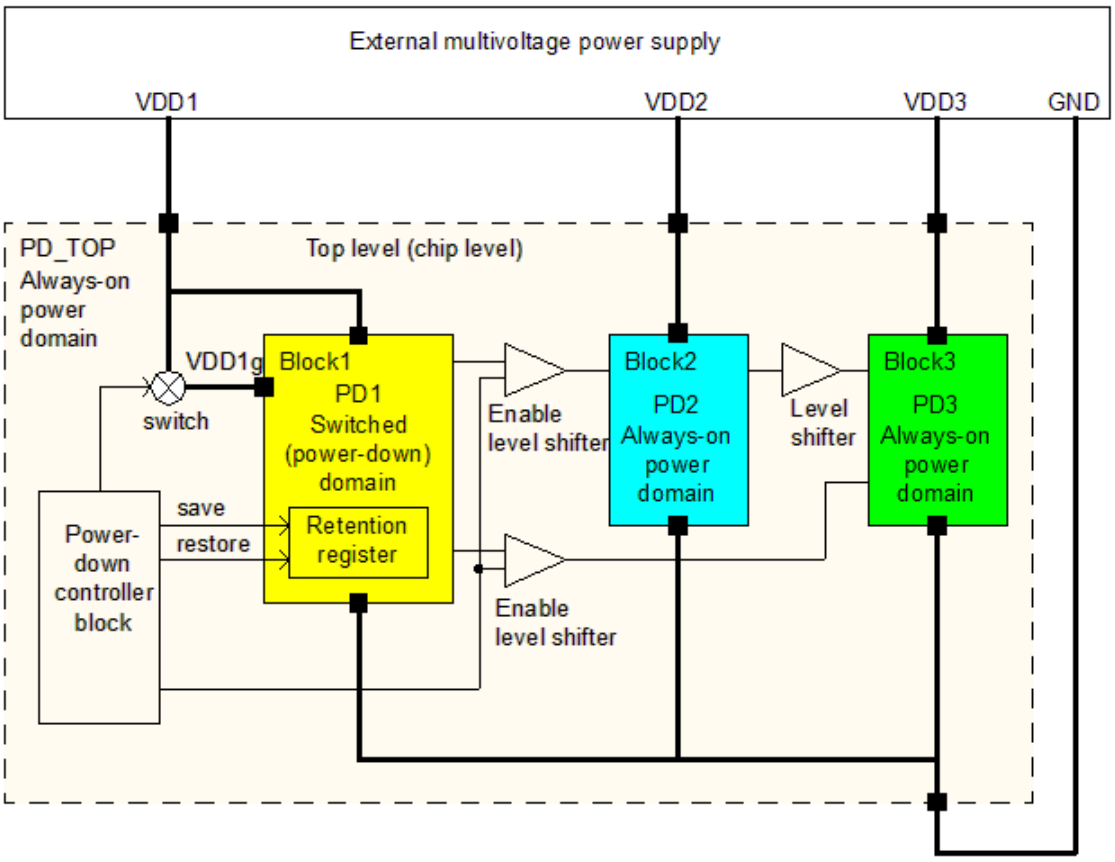
In a power domain that has power switching, any registers that must retain data during shutdown can be implemented as *retention registers*. A retention register has a separate, always-on supply net, sometimes called the backup supply, which keeps the data stable in the retention register while the primary supply of the domain is shut down.

Power Network Examples

The power network example shown in Figure 21 demonstrates some of the power intent concepts. This chip is designed to operate with three power supplies that are always on (although the UPF syntax also supports externally switchable power supplies), at three different voltage levels. The top-level chip occupies the top-level power domain, PD_TOP. The domain PD_TOP is defined to have four supply ports: VDD1, VDD2, VDD3, and GND. The black squares along the border of the power domain represent the supply ports of that domain. Note that this diagram shows the connections between power domains and is not meant to represent the physical layout of the chip.

Feedback

Figure 21 Power Intent Specification Example



In addition to the top-level power domain, PD_TOP, there are three more power domains defined, called PD1, PD2, and PD3, created at the levels of three hierarchical blocks, Block1, Block2, and Block3, respectively. Each block has supply ports (shown as black squares in the diagram) to allow supply nets to cross from the top level down into the block level.

In this example, PD_TOP, PD2, and PD3 are always-on power domains that operate at different supply voltages, VDD1, VDD2, and VDD3, respectively. PD1 is a power domain that has two supplies: a switchable supply called VDD1g and an always-on supply from VDD1. The always-on power supply maintains the domain's retention registers while VDD1g is powered down.

A power switch shuts off and turns on the power net VDD1g, either by connecting or disconnecting VDD1 and VDD1g. A power-down controller logic block at the top level generates the control signal for the switch. It also generates the save and restore signals for the retention registers in domain PD1 and the control signals for the isolation cells

Feedback

between domain PD1 and the always-on domains PD2 and PD3. These isolation cells generate known signals during times that VDD1g is powered down.

Because domains PD1, PD2, and PD3 operate at different supply voltages, a level shifter must be present where a signal leaves one of these domains and enters another. In the case of the signals leaving PD1 and entering PD2 or PD3, the interface cells must be able to perform both level shifting and isolation functions, because PD1 can be powered down.

To access a particular power domain, supply port, or supply net in the design hierarchy, you specify the hierarchical path in the design to the scope where the object exists, and end with the object name; or you can change the scope of the tool to that hierarchical level and specify the object name directly. For example, suppose you want to create a supply net called VDD1 and a supply port called PRT1 at the scope of Block1, which is in power domain PD1, and you want to make a connection from the net to the port within the scope of Block1. From the scope of the top level (the default scope), you could use the following commands:

```
create_supply_net VDD1 -domain Block1/PD1
create_supply_port PRT1 -domain Block1/PD1
connect_supply_net Block1/VDD1 -ports {Block1/PRT1}
```

An alternative command entry method is to change the scope to Block1 using either the `set_scope` or `current_instance` command and using simple (not hierarchical) names in the commands:

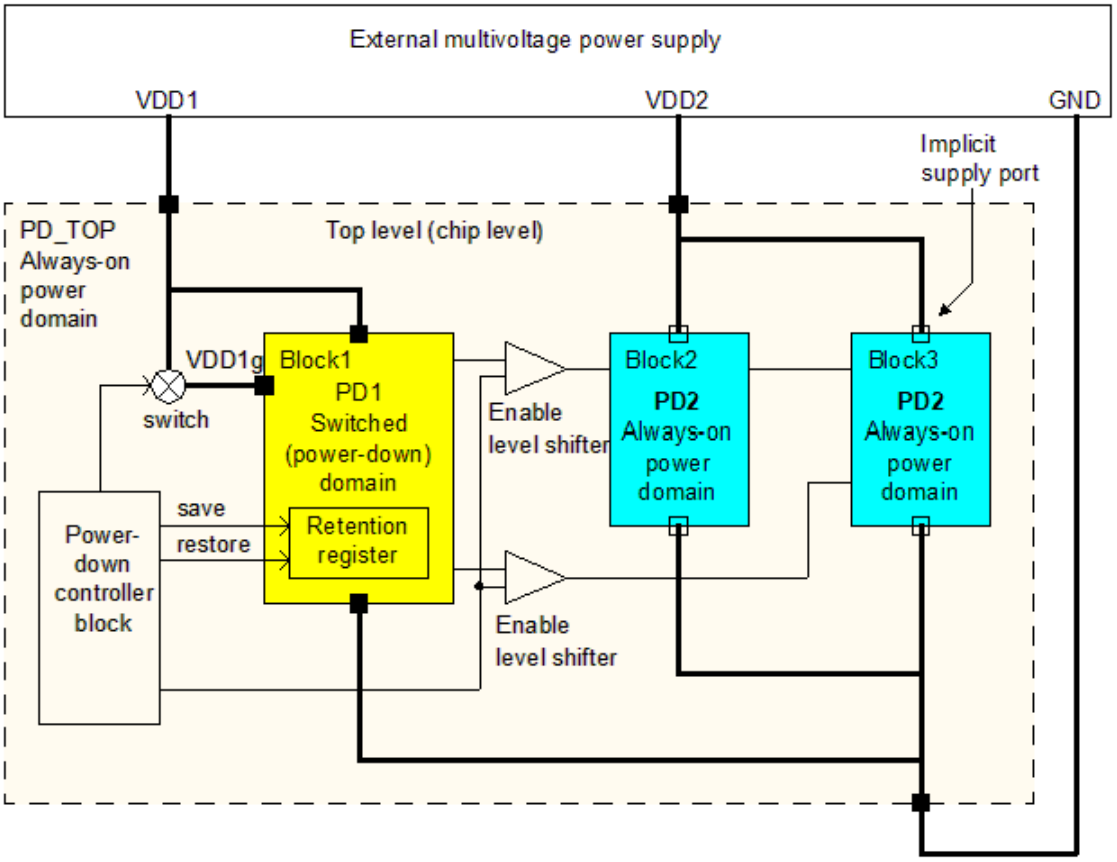
```
set_scope Block1
create_supply_net VDD1 -domain PD1
create_supply_port PRT1 -domain PD1
connect_supply_net VDD1 -ports {PRT1}
set_scope ...
```

In the foregoing power intent strategy, each hierarchical block is assigned to its own power domain. It is also possible for a hierarchical block to belong to the same power domain as its parent or for multiple hierarchical blocks to belong to a single power domain.

In the alternative power intent specification shown in Figure 22, Block2 and Block3 share the same power characteristics, so they are assigned to a single power domain called PD2, created at the top level of hierarchy. The two blocks can use the supply nets defined at the top level, so it is not necessary to create the supply ports explicitly between the lower-level blocks and the current level of hierarchy.

Feedback

Figure 22 Alternative Power Intent Specification



The scope (hierarchical level) of power domain PD1 is the block Block1, whereas the scope of power domain PD2 is the top level. Block1 uses the supply nets within the scope of Block1, whereas Block2 and Block3 use the supply nets within the scope of the top level. The supply net VDD1 crosses from the top level of the design down to the level of Block1, so it must pass through a supply port defined at the scope of Block1, and the supply net is said to be reused in domain PD1. The supply net VDD2 is defined at the top level, in power domain PD_TOP, but it is also used in power domain PD2, so it is said to be reused in PD2. A reused supply net spans different domains and has the same name in these domains.

To define the power domain strategy shown in the first example (Figure 21), you would use the following commands:

```
create_power_domain PD_TOP
create_power_domain PD1 -elements {Block1} -scope Block1
create_power_domain PD2 -elements {Block2} -scope Block2
create_power_domain PD3 -elements {Block3} -scope Block3
```

Feedback

To define the power domain strategy shown in the second exampe1 (Figure 22), with Block1 and Block2 in the same domain, you would use the following commands:

```
create_power_domain PD_TOP
create_power_domain PD1 -elements {Block1} -scope Block1
create_power_domain PD2 -elements {Block2 Block3}
```

or equivalently:

```
create_power_domain PD_TOP
set_scope Block1
create_power_domain PD1
set_scope ...
create_power_domain PD2 -elements {Block2 Block3}
```

Even if Block2 and Block3 share the same power supply characteristics, you might want to place them in separate power domains anyway. Doing so would cause the synthesis and implementation tools to implement the power supply connections to the blocks separately. For example, if the blocks are placed in power-down domains, the physical implementation tool might use larger or faster switching cells to provide power to the block that draws a larger current.

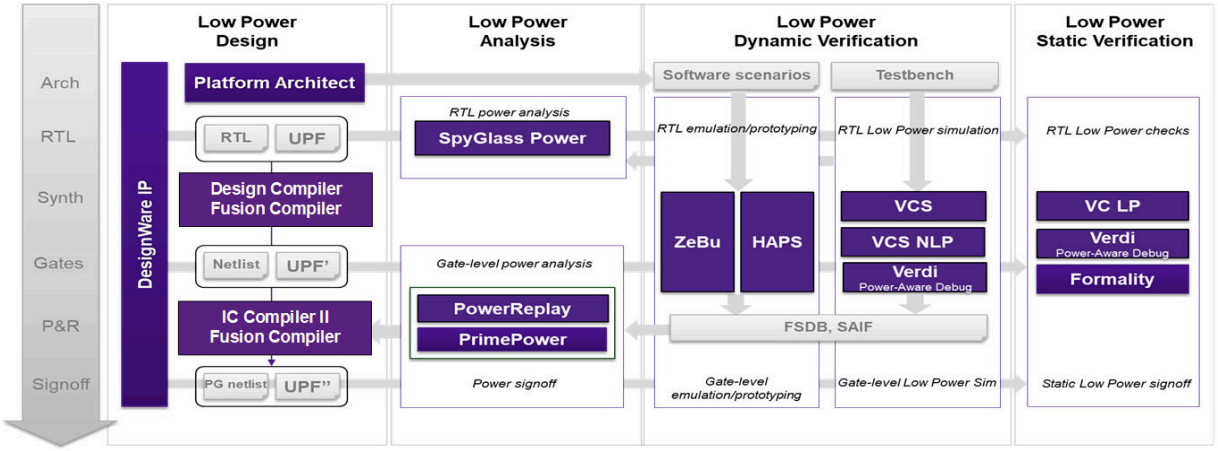
Bidirectional signals cannot be isolated or level-shifted. If any bidirectional signals require isolation or level shifting at a power domain boundary, modify the design to separate those signals into distinct input signals and output signals.

Synopsys Multivoltage Flow

The Synopsys multivoltage synthesis, implementation, and verification flow is shown in Figure 23. The flow starts with the register-transfer level (RTL) description of the logic of the design, together with a separate UPF description of the power intent of the design. The RTL and UPF descriptions are contained in separate files so that they can be maintained and modified separately. The initial UPF description is designated the UPF0 file in this example.

Feedback

Figure 23 Multivoltage Flow With Synopsys Tools



The Design Compiler tool reads in the RTL logic and original UPF power intent descriptions, and based on their contents, synthesizes a gate-level netlist and an updated UPF file, designated UPF' (UPF prime) in this example. The UPF' file contains the original UPF information plus explicit supply net connections for special cells created during synthesis.

The IC Compiler, IC Compiler II, or Fusion Compiler tool reads in the gate-level netlist and UPF' power description files, and based on the file contents, performs physical implementation (placement and routing), producing a modified gate-level netlist, a complete power and ground (PG) netlist, and an updated UPF file, UPF' ' (UPF double-prime). The UPF' ' file contains the UPF' information plus any modifications to low-power circuit structures resulting from physical implementation, such a power switches.

The data files used in this flow can be used for functional verification with the VCS simulator, formal equivalence checking with the Formality tool, and timing and power verification with PrimeTime, PrimePower, and PrimeRail tools.

The VCS NLP multivoltage simulation tool can be used for functional verification of the design with multivoltage features at several different stages of the flow: at the RTL level before synthesis, at the gate level after synthesis with power-related cells added, and after placement and routing with the power switches added. At each level, VCS NLP simulates the design to verify the impact of voltage changes in a power-managed chip, allowing you to accurately and reliably detect any low-power design issues. The VC LP tool checks for adherence to multivoltage rules and reports any problems related to power connectivity, power architecture, or power intent consistency.

The PrimeTime tool reads the gate-level netlist from the synthesis or physical implementation tool and also reads the UPF descriptions generated by those tools. It uses

Feedback

the UPF information to build a virtual model of the power network and to annotate voltage values appropriately on each power pin of each leaf-level gate instance in the design. The PrimeTime tool does not modify the power domain description in any structural or functional way, so it does not write out any UPF commands.

UPF Commands Supported by Synopsys Tools

Synopsys tools currently support a large subset of the commands in the IEEE 1801 (UPF) standard. In addition, they support some UPF-like power intent commands that are not part of the official standard. Certain UPF commands are supported by some Synopsys tools but not others. For example, the synthesis and physical implementation tools generate new UPF commands, so they support the `save_upf` command, whereas the static timing and verification tools do not generate any UPF commands, so they do not support the `save_upf` command.

Table 2 lists the major UPF commands and shows whether they are supported by the following Synopsys tools: Synthesis tools (Design Compiler, Power Compiler, and DFT Compiler), IC Compiler, IC Compiler II, Fusion Compiler, PrimeTime and PrimePower, Formality, PrimeRail, VCS NLP, and VC LP. This is the table entry key:

- yes command supported and used by tool
- ok command accepted as valid but not used by tool
- -- command not supported

Table 2 Support for UPF Commands in Synopsys Tools

UPF command	Synthesis	IC Compiler	IC Compiler II	Fusion Compiler	Prime Time	Formality	Prime Rail	VCS NLP	VCS LP
Basic power network:									
create_power_domain	yes	yes	yes	yes	yes	yes	yes	yes	yes
create_supply_port	yes	yes	yes	yes	yes	yes	yes	yes	yes
create_supply_net	yes	yes	yes	yes	yes	yes	yes	yes	yes
connect_supply_net	yes	yes	yes	yes	yes	yes	yes	yes	yes
set_domain_supply_net	yes	yes	yes	yes	yes	yes	yes	yes	yes
create_power_switch	yes	yes	yes	yes	yes	yes	yes	yes	yes
map_power_switch	yes	yes	yes	yes	ok	ok	ok	ok	yes
create_supply_set	yes	yes	yes	yes	yes	yes	yes	yes	yes
associate_supply_set	yes	yes	yes	yes	yes	yes	yes	yes	yes
set_equivalent	yes	yes	yes	yes	yes	yes	yes	yes	yes

Feedback

Table 2 Support for UPF Commands in Synopsys Tools (Continued)

UPF command	Synthesis	IC Compiler	IC Compiler II	Fusion Compiler	Prime Time	Formality	Prime Rail	VCS NLP	VCS LP
Level shifter:									
set_level_shifter	yes	yes	yes	yes	ok	ok	ok	ok	yes
map_level_shifter_cell	yes	yes	yes	yes	ok	ok	ok	ok	yes
name_format	yes	yes	yes	yes	ok	ok	ok	yes	yes
use_interface_cell	yes	--	yes	yes	yes	yes	--	--	yes
Isolation:									
set_isolation	yes	yes	yes	yes	yes	yes	yes	yes	yes
set_isolation_control	yes	yes	yes	yes	yes	yes	yes	yes	yes
map_isolation_cell	yes	yes	yes	yes	ok	ok	ok	ok	yes
name_format	yes	yes	yes	yes	ok	ok	ok	yes	yes
use_interface_cell	yes	--	yes	yes	yes	yes	--	--	yes
Retention:									
set_retention	yes	yes	yes	yes	yes	yes	yes	yes	yes
set_retention_control	yes	yes	yes	yes	yes	yes	yes	yes	yes
map_retention_cell	yes	yes	yes	yes	ok	ok	ok	ok	yes
set_retention_elements	yes	--	--	--	--	--	--	yes	yes
Power models:									
add_parameter	yes	--	yes	yes	--	--	--	yes	--
apply_power_model	yes	--	yes	yes	--	--	--	yes	--
define_power_model	yes	--	yes	yes	--	--	--	--	--
Power states:									
create_pst	yes	yes	yes	yes	ok	yes	ok	yes	yes
add_port_state	yes	yes	yes	yes	ok	yes	ok	yes	yes
add_pst_state	yes	yes	yes	yes	ok	yes	ok	yes	yes
add_power_state	yes	yes	yes	yes	ok	yes	ok	yes	yes
create_power_state_group	yes	yes	yes	yes	yes	yes	yes	yes	yes
add_state_transition	yes	--	yes	yes	--	--	--	yes	ok
describe_state_transition	yes	--	yes	yes	--	--	--	yes	yes
Logic editing:									
create_logic_net	yes	yes	yes	yes	ok	yes	ok	yes	yes
create_logic_port	yes	yes	yes	yes	ok	yes	ok	yes	yes
connect_logic_net	yes	yes	yes	yes	ok	yes	ok	yes	yes

Feedback

Table 2 Support for UPF Commands in Synopsys Tools (Continued)

UPF command	Synthesis	IC Compiler	IC Compiler II	Fusion Compiler	Prime Time	Formality	Prime Rail	VCS NLP	VCS LP
Utility:									
load_upf	yes	yes	yes	yes	yes	yes	yes	yes	yes
save_upf	yes	yes	yes	yes	--	ok	--	--	--
set_scope	yes	yes	yes	yes	yes	yes	yes	yes	yes
set_design_attributes	yes	yes	yes	yes	ok	yes	ok	yes	yes
set_port_attributes	yes	yes	yes	yes	yes	yes	yes	yes	yes
set_repeater	--	--	yes	yes	--	yes	--	yes	yes
Query:									
find_objects	yes	yes	yes	yes	yes	yes	yes	yes	yes
query_cell_instances	yes	yes	--	--	yes	--	yes	yes	--
query_cell_mapped	yes	yes	--	--	yes	--	yes	yes	--
query_map_power_switch	yes	yes	--	--	--	--	--	--	--
query_net_ports	yes	yes	--	--	yes	--	yes	ok	--
query_port_net	yes	yes	--	--	yes	--	yes	ok	--
query_port_state	yes	yes	--	--	--	--	--	--	--
query_power_switch	yes	yes	--	--	--	--	--	yes	--
query_pst	yes	yes	--	--	--	--	--	yes	--
query_pst_state	yes	yes	--	--	--	--	--	yes	--
Simulation/verification extension:	ok	ok	ok	ok	yes	ok	yes	yes	ok
set_design_top	ok	ok	ok	ok	ok	yes	ok	yes	ok
set_partial_on_translation	ok	--	ok	ok	--	--	--	yes	yes
sim_assertion_control									
VCS NLP / VC LP extension:	--	--	--	--	--	--	--	yes	ok
bind_checker	--	--	--	--	--	--	--	yes	ok
describe_state_transition	--	--	--	--	--	--	--	yes	yes
set_retention_elements	--	--	--	--	--	--	--	ok	ok
set_simstate_behavior	--	--	--	--	--	--	--	yes	--
query_isolation	--	--	--	--	--	--	--	yes	--
query_power_domain	--	--	--	--	--	--	--	yes	--
query_retention	--	--	--	--	--	--	--	yes	--
query_retention_control	--	--	--	--	--	--	--	yes	--
query_supply_net									

The following subsections describe the UPF commands supported by Synopsys tools. These descriptions are intended to provide an overview of how the commands work in the

Feedback

Synopsys flow. They are not intended to serve as a comprehensive command reference. Only the commands and command options supported by Synopsys tools are shown. The official IEEE 1801 (UPF) specification has some additional commands and command options that are not supported by Synopsys tools. Conversely, some Synopsys commands offer features that are not a part of the IEEE 1801 (UPF) specification.

For more information about using the UPF commands, see the IEEE 1801 (UPF) specification, the Synopsys tool user guides, or the man pages for the commands.

For more information about Synopsys tool support for individual command options, see the SolvNet article 021264, "Supported Subset for the IEEE 1801 Unified Power Format (UPF) Standard."

Basic Power Network Commands

The basic power commands define the power domains of the design and the supply ports, supply nets, and power switches of each domain.

create_power_domain

```
create_power_domain domain_name
  [-elements list]
  [-supply {supply_set_handle supply_set_name}]
  [-include_scope]
  [-scope instance_name]
  [-update]
```

The `create_power_domain` command defines a power supply distribution network at the current scope (hierarchical level) or at the scope of a specified hierarchical instance. A power domain must have one primary supply net and one primary ground net, and can optionally have additional supply nets, supply ports, and power switches.

The `-elements` option specifies a list of design elements that are assigned to the power domain (the extent of the power domain). The `-elements` option supports the `{.}` list as follows:

```
create_power_domain PD_TOP -elements {.}
```

If the current domain is TOP, then the TOP domain and all the hierarchy beneath it are part of the PD_TOP domain. You cannot use the `-scope` and `-elements {.}` options together.

For the IC Compiler II and Fusion Compiler tools, the `-elements` option's list can also refer to instance names, named blocks, and signals for `create_power_domain` commands that are specified inside a power model of a hard macro. Named blocks or signal names can only be specified using their simple names. Hierarchical names are not allowed.

The `-include_scope` option causes the entire hierarchical level of the domain to be included in the extent of the domain. If neither `-elements` nor `-include_scope` is used,

Feedback

the power domain includes all elements in the current scope (including lower-level elements) not already assigned to power domains by previous `create_power_domain` commands.

The `-supply` option specifies a supply set, to be defined and associated with the power domain. This option can be used multiple times, to define multiple supply sets that are associated with the power domain. The supported supply set handles are `primary`, `default_retention`, `default_isolation`, and `extra_supplies_#`. For more details on creating supply sets, see create_supply_set.

Every design must have a top-level power domain, created in the top-level scope with the `-include_scope` option (to include everything at the top level) or without any command options (to include all elements at the top level not already assigned to power domains).

The `-scope` option specifies the name of an instance in which to create the power domain; it defines the domain boundary within the logic design. Specifying a scope has the same effect as changing the tool to the level of instance with the `set_scope` or `current_instance` command, creating the power domain at that level, and then changing back to the original hierarchical level. If you do not specify a scope, the power domain is created at the level of the current scope.

The `-update` option updates the supply set association of an existing power domain. This option lets you:

- Specify a new supply set handle, and associate it with a supply set
- Specify a new supply set handle, without associating it with a supply set
- Associate an existing supply set handle to a supply set

The `-update` option can be used only with the `-supply` option. It is an error to use the `-update` option along with any other option or to update a supply set handle that is already associated with a supply set.

A design element in the extent of a power domain can be an instance of a hierarchical or leaf-level cell. However, the synthesis and physical implementation tools only allow hierarchical and macro cells (not leaf-level cells) in the `-elements` list. A macro cell is a black box cell, with functionality defined beyond the scope of the synthesis tool, so it does not require any mapping and optimization. If you want a set of leaf-level cells to belong to a power domain other than the top-level power domain, they must be grouped within a hierarchical cell.

The cells in the `-elements` list are the root cells of the power domain. These cells must lie within the scope (level of hierarchy) where the power domain is created. If the definition of a power domain uses the `-include_scope` option and the power domain is created in a hierarchical cell by using `-scope` *`instance_name`*, the hierarchical cell is considered the root cell of the power domain.

Feedback

A more specific power domain assignment overrides a more general assignment. For example, if you assign BlockA to power domain PD1, and BlockA contains two lower-level blocks A1 and A2, the lower-level blocks are also assigned to domain PD1. However, you can later assign block A1 to a new power domain PDA1, which overrides the previous assignment of block A1 to domain PD1. Thus, the power domain membership of each cell in the design is determined by the following rules: if the cell is a root cell of a power domain, it belongs to that power domain; otherwise, if the cell has a parent cell, it belongs to the same power domain as its parent cell; otherwise, the cell belongs to the top-level power domain.

Any given cell can be assigned to no more than one power domain. To fully specify the power intent of a design, every cell in the design must belong to exactly one power domain.

create_supply_port

```
create_supply_port port_name
  [-domain domain_name]
  [-direction in | out]
```

The `create_supply_port` command creates a supply port at the current scope or at the scope of a power domain specified with the `-domain` option. A supply port is a power supply connection point between the current scope and the next higher-level (parent) scope. A supply port allows a supply net to cross from one hierarchical level to another.

If `-domain` is used, the port is created at the level of the specified domain, and the specified port name must be a simple (not hierarchical) name. In that case, the supply port is created at the scope where the specified power domain exists. For example,

```
prompt> create_supply_port VDD1P -domain Block1/PD1
```

If `-domain` is not used, the specified port name must be a full hierarchical name consisting of the instance path, starting from the current level, down to the scope where the supply port is to be added. For example,

```
prompt> create_supply_port Block1/VDD1P
```

The `-direction` option specifies the direction that power state information (on or off state) is propagated through the supply network connected to the port, either `in` or `out`. For an input supply port, the state information of the external supply net is propagated down from the parent scope into the scope of the supply port. For an output port, the state information of the internal supply net is propagated up from the scope of the supply port into the parent scope. The default direction is `in`.

create_supply_net

```
create_supply_net net_name
  [-domain domain_name]
  [-reuse]
```

```
[-resolve unresolved | parallel | one_hot | parallel_one_hot |
          user_defined_resolution_function]
```

The `create_supply_net` command creates a supply net to supply power or ground to a power domain. You must specify the name for the new supply net and the name of the existing power domain in which the supply net will be used. The supply net is specified as a simple (not hierarchical) name. The domain can be specified as a simple name (for example, PD1), which creates a supply net at the level of the current scope; or it can be a hierarchical name (for example, Block1/PD1), which creates a supply net at the specified scope.

The supply net is propagated through implicitly created ports and nets throughout the logic hierarchy, starting from the level where it is created and downward through the hierarchy of all elements that belong to the same power domain. For example, if the domain has a hierarchical block containing three levels of lower-level blocks, creating the supply net implicitly creates supply ports between the parent block and children blocks and between the children blocks and the grandchildren blocks. These supply ports propagate the supply net throughout the hierarchy of the parent block.

The `-reuse` option causes an existing supply net outside of the specified power domain to be reused and extended into that domain. This does not create a new supply net, but extends an existing supply net to span multiple domains.

The `-resolve` option specifies how the state and voltage of a supply net are resolved when the supply net is driven by one or more power switches. You can specify `unresolved` (single driver allowed), `parallel` (multiple drivers allowed), `one_hot` (no more than one driver at a time), or `parallel_one_hot` (no more than one driver at a time with multiple driver supply paths).

Synopsys tools support the usage of a custom resolution function defined in a package used by the VCS-NLP tool. A custom resolution function implies that multiple drivers on a net are allowed. Tools in the flow that do not have access to the custom resolution function can read and save the function name in the UPF and check for conflicting functions applied to the same net, but otherwise do not use the function information.

A supply net object is an abstract representation of a power or ground net in the design. It is not a true netlist object, but rather a piece of design data that acts as a repository for information about a particular power or ground net, including the list of supply ports and cell pins that need to be connected together.

connect_supply_net

```
connect_supply_net supply_net_name
  -ports list
  [-vct vct_name]
```

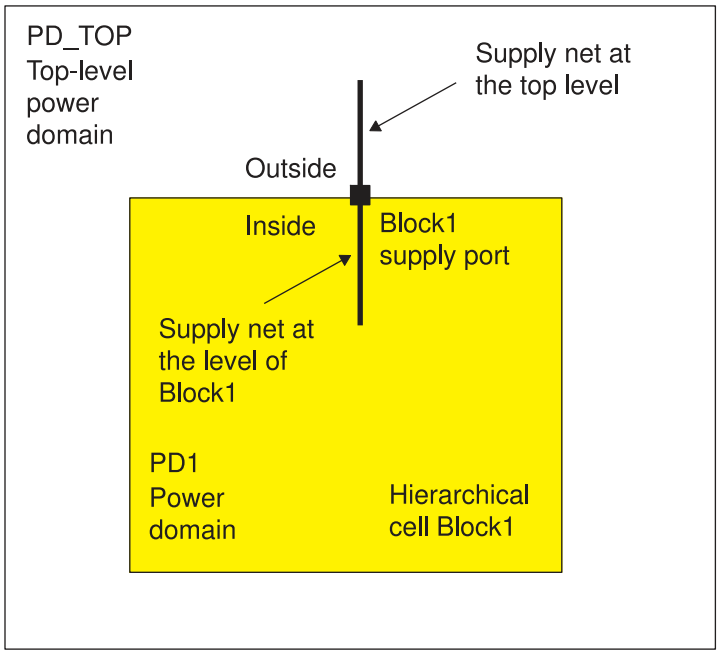
The `connect_supply_net` command specifies an explicit connection of a supply net to a list of supply ports, thereby overriding any implicit connections that might otherwise apply.

Feedback

The command specifies the name of an existing supply net and lists the supply ports to be connected to the supply net.

Each supply port has two ends: an inside end and an outside end. A supply net from the logical hierarchy above a supply port can connect only to the outside end of the supply port, and a supply net at the scope of the supply port can connect only to the inside end of the supply port, as shown in Figure 24. Supply nets existing below the logical hierarchy of the supply port can connect to the supply port only through supply pins at each hierarchical level.

Figure 24 Supply Net Connections to the Two Ends of a Supply Port



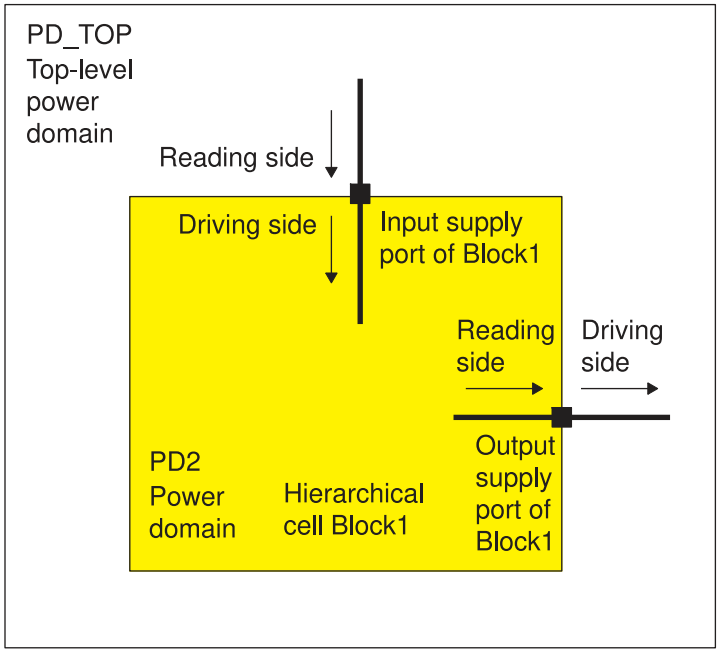
A supply port can be connected only one time at each end. This restriction ensures a unique mapping from a net segment in the PG netlist to the corresponding supply net. To make the outside and inside connections, with the current scope set to the top level, you could use commands similar to the following:

```
create_supply_port VDD1P -domain Block1/PD1
create_supply_net VDD1 -domain Block1/PD1
connect_supply_net Block1/VDD1 -ports Block1/VDD1P
connect_supply_net VDD1 -ports Block1/VDD1P
```

VCS and Formality enforce restrictions on the “direction” of power flow. An output port “drives” its external net and “reads” its internal net; the “driving side” is the external net and

the “reading side” is the internal net. On the other hand, an input port “drives” its internal net and “reads” its external net; the “driving side” is the internal net and the “reading side” is the external net. See Figure 25. You cannot connect more than one driver to a supply net or reuse an existing net with multiple drivers, unless that net has a resolution mechanism.

Figure 25 Power Flow Direction



set_domain_supply_net

```
set_domain_supply_net domain_name
  -primary_power_net supply_net_name
  -primary_ground_net supply_net_name
```

The `set_domain_supply_net` command specifies the primary power net and primary ground net for an existing power domain. Every power domain must have these supply nets defined. They are the default power nets connected to the logic elements (or inferred cells) of the power domain. At the gate level, the power and ground pins of inferred gates are connected to the primary power and ground nets unless specified otherwise by the `connect_supply_net`, `set_retention`, or `set_isolation` command. The two supply nets must already exist within the scope of the power domain.

If the domain and supply nets are not within the current scope, you can specify the names hierarchically. For example,

```
set_domain_supply_net Block1/PD1 \
  -primary_power_net Block1/VDD1 \
  -primary_ground_net Block1/GND
```

create_power_switch

```
create_power_switch switch_name
  -domain domain_name
  -output_supply_port {port_name supply_net_name}
  {-input_supply_port {port_name supply_net_name}}*
  {-control_port {port_name net_name}}*
  {-on_state {state_name input_supply_port {boolean_function}}}*
  [-ack_port {port_name net_name [{boolean_function}]}]*
  [-ack_delay {port_name delay}]*
  [-off_state {state_name {boolean_function}}]*
  [-on_partial_state {state_name {boolean_function}}]*
  [-error_state {state_name {boolean_function}}]*
```

*Asterisk indicates that the item in curly or square braces can be repeated in the command.

The `create_power_switch` command creates an instance of a power switch in the scope of a power domain. The switch has at least one input supply port and one output supply port. When the switch is on, it connects the input supply port (or one of multiple input supply ports) to the output supply port. When the switch is off, the output supply port is shut down and has no power.

The command must specify the power domain in which to create the power switch, the output supply port, the input supply port, the control port or ports, and the “on” state of the switch (a Boolean function). It can optionally specify the characteristics of the acknowledge port or ports. The ports specified in the command are the connections points of the power switch. The `-on_state`, `-ack_delay`, and `-off_state` settings are used only for simulation, not synthesis or implementation.

The state of the switch depends on one or more input control ports defined for the switch. The switch is on if the value on the control ports matches the `-on_state` Boolean expression. Otherwise, the switch is off. One or more “off” states can be identified as error states by the `-off_state` option for checking by the simulator. The simulation semantics for these error states is tool-dependent.

A power switch can optionally have one or more acknowledge output ports that indicate the state of the switch. If a Boolean function is specified by the `-ack_port` option, the result of the Boolean function is driven on the acknowledge port. Otherwise, the value 1 or 0 is driven on the acknowledge port when the switch is turned on or off, respectively, with a delay specified by the `-ack_delay` option. If the delay is specified as an integer, the time

unit is interpreted as the precision of the simulator. The default delay value is zero. Any `-ack_port` or `-on_state` Boolean expressions are written in SystemVerilog syntax.

The Design Compiler tool does not use the information in the `create_power_switch` command. Although it records the `-ack_port`, `-on_state`, `-on_partial_state`, `-off_state`, and `-error_state` settings, it does not use this information for synthesis. Additional non-UPF commands might be required in the physical implementation tool to specify the mapping. Because verification tools such as VCS do not support any mapping commands, they use the information embedded in the `-ack_port`, `-on_state`, `-on_partial_state`, `-off_state`, and `-error_state` settings to determine functionality.

From the implementation side, the Design Compiler tool does not create any placeholder for the power switch in the generated netlist. Instead, it passes on any power switch commands received in its input UPF to its output UPF. Connections between the power ports on the abstract switch and the supply nets are entirely captured by the `create_power_switch` command. No `connect_supply_net` command can refer to the power ports on the abstract switch. Other commands, such as `add_port_state` and `create_pst`, can refer to the supply ports by hierarchical names.

The following is a simple power switch definition:

```
create_power_switch SW1 \
  -domain PD_TOP \
  -output_supply_port {SWOUT VDD1g} \
  -input_supply_port {SWIN1 VDD1} \
  -control_port {CTRL swctl} \
  -on_state {ON VDD1 {!swctl}}
```

The following is a more complex example that includes two input supply ports, three input control ports, four different “on” states, and an error state:

```
create_power_switch sw1 \
  -domain PD_SODIUM \
  -output_supply_port {vout VN3} \
  -input_supply_port {vin1 VN1} \
  -input_supply_port {vin2 VN2} \
  -control_port {ctrl_small ON1} \
  -control_port {ctrl_large ON2} \
  -control_port {ss SUPPLY_SELECT} \
  -on_state {partial_s1 vin1 {ctrl_small & !ctrl_large & ss}} \
  -on_state {full_s1 vin1 {ctrl_small & ctrl_large & ss}} \
  -on_state {partial_s2 vin2 {ctrl_small & !ctrl_large & !ss}} \
  -on_state {full_s2 vin2 {ctrl_small & ctrl_large & !ss}} \
  -error_state {no_small {!ctrl_small & ctrl_large}}
```

map_power_switch

```
map_power_switch switch_name
  -domain domain_name
  -lib_cells list
```

The `map_power_switch` command can be used to explicitly specify which power switch cell is to be used to implement a specified switch instance. The command must specify the name of an existing switch previously created with the `create_power_switch` command, the name of the power domain containing the switch, and the list of library cells that can be used to implement the switch.

The Design Compiler tool does not use the information in this command, but simply passes the command to the output UPF. The physical implementation tool is solely responsible for using the command to select the library cells for implementing power switches.

For a single-input, single-output, single-control switch, the power pin names and signal pin names in the library cells need not match the virtual port names declared in the `create_power_switch` command. However, for a multiple-control switch, additional non-UPF commands might be needed in the physical implementation tool to perform sophisticated tasks such as mapping to a daisy chain of switches. After mapping, the physical implementation tool might generate additional `connect_supply_net` commands in its output UPF to represent the connections between the supply nets and the power pins on the mapped switches.

create_supply_set

```
create_supply_set supply_set_name
  [-function {function net_name}]
  [-update]
```

The `create_supply_set` command creates a supply set at the current level of logic hierarchy. A supply set is a collection of supply nets. A supply set can be considered as a unified and progressively defined bundle of supply nets that is not specific to a particular power domain. A supply set can be used within the scope where it is defined and in scope under it.

The supply set concept eliminates the need to define the supply nets and ports in the design in the synthesis tool. The supply sets must be associated with the supply nets and ports before performing physical synthesis.

A supply set consists of the following two functions:

- Power
- Ground

The `-function` option specifies the functionality to which the net name should be associated. The option takes two parameters. The first is the name of the function, either power or ground. The second is the name of the supply net that is in the scope of the supply set.

The functions of the supply set can be accessed by using the name of the supply set and the name of the function. To access the power function of a supply set, specify

Feedback

supply_set_name.power. To access the ground function of a supply set, specify supply_set_name.ground.

The -update option causes the tool to associate existing supply nets to the power and ground functions of the existing supply set. Only one update is allowed for each function of the supply set.

Supply sets are independent of the domain in which they are created. They can only be updated with domain-independent nets. The domain-independent supply nets must be created in the same scope as the supply set. To restrict the supply set available for optimization in a power domain, use the extra_supplies_# keyword (extra_supplies_ followed by a number suffix) with the -supply option of the create_power_domain command. The extra_supplies keyword can also be used without the number suffix. In this case, the power domain does not use supply nets other than the nets that are already defined with the UPF strategies.

By default, the power domain uses supply nets defined in the power domain or the domain-independent supply nets. When power domains contain supply sets with the extra_supplies_# keyword, the power domain is restricted to use only the following types of power supplies:

- Primary supply
- Supplies specified in the isolation strategies
- Supplies specified in the retention strategies
- Domain-dependent supplies defined or reused

Feedback

associate_supply_set

```
associate_supply_set supply_set_name
  -handle supply_set_name
```

The `associate_supply_set` command associates a supply set handle to another supply set. When a supply set handle is associated with a supply set, the tool considers the two supply sets to be connected and are resolved to the same supply net or the same pair of PG pins. The `supply_set_name` specifies the name of the supply set that must be associated with the handle specified with the `-handle` option.

You can associate a supply set with another supply set handle only when the following conditions are met.

- Associating a supply set handle to a supply set can be done only one time.
- Associating a supply set handle to a supply set must not cause circular associations
- User-defined supply sets cannot be specified with the `-handle` option of the `associate_supply_set` command.
- The supply set handle specified with the `-handle` option of the `associate_supply_set` command must be at the same or below the scope of the specified supply set.
- While associating a supply set handle to a supply set, the supply set must be available in the power domain where the supply set is available.

Feedback

set_equivalent

```
set_equivalent -nets supply_nets | -sets supply_sets
  [-function_only]
```

The `set_equivalent` command declares a list of supply nets or supply sets as equivalent, meaning that members always share the same state and voltage, whether or not they are electrically shorted together explicitly in the design database. For example,

```
prompt> set_equivalent -nets {VDDa VDDb VDDc}  # equivalent supply nets

prompt> set_equivalent -sets {SS1a SS1b}       # equivalent supply sets
```

The `-function_only` option specifies that the supply nets or supply sets are either electrically equivalent or have independent and parallel circuitry, not necessarily shorted together.

Level Shifter Commands

The level shifter commands let you specify the strategy for inserting level shifters between power domains operating at different voltages.

Feedback

set_level_shifter

```
set_level_shifter strategy_name
  -domain domain_name
  [-elements port_pin_list]
  [-exclude_elements exclude_list]
  [-applies_to inputs | outputs | both]
  [-applies_to_boundary upper | lower | both]
  [-threshold float]
  [-rule low_to_high | high_to_low | both]
  [-location self | parent | fanout | automatic]
  [-no_shift]
  [-force_shift]
  [-name_prefix prefix_string]
  [-name_suffix suffix_string]
  [-update]
  [-source source_supply]
  [-sink sink_supply]
```

The `set_level_shifter` command can be used to set a strategy for inserting level shifters during implementation. The synthesis and implementation tools place level shifters on signals that have sources and sinks operating at different voltages, following the specified strategy. If a level shifter strategy is not specified on a particular power domain, the default level shifter strategy applies to all elements in the power domain, using the default strategy settings.

The IC Compiler II and Fusion Compiler tools insert a level shifter only where there is a voltage difference and the situation matches the level shifter strategy specified in the UPF file. If there is no level shifter strategy specified, you can create a new one at any time by using the `create_mv_cells -generate_strategy level_shifters` command. A strategy specified with this command is included in the UPF written out by the `save_upf` command in the IC Compiler II or Fusion Compiler tool.

The `-elements` option specifies a list of ports and pins in the domain to which the strategy applies, overriding any `-threshold` or `-rule` settings.

The `-exclude_elements` option specifies a set of ports to which this strategy does not apply. Both the `-elements` and `-exclude_elements` options support the `{.}` as a valid element list. The `{.}` represents all instances in the current scope.

The `-threshold` option defines how large the voltage difference must be between the driver and sink before level shifters are inserted, overriding any such specification in the cell library. The `-rule` option can be set to `low_to_high`, `high_to_low`, or `both`. If `low_to_high` is specified, signals going from a lower voltage to a higher voltage get a level shifter when the voltage difference exceeds the `-threshold` value. Similarly, if `high_to_low` is specified, signals going from a higher voltage to a lower voltage get a level shifter when the voltage difference exceeds the `-threshold` value. The default behavior is `both`, which means that a level shifter is inserted in either situation.

Feedback

The `-location` option specifies where the level-shifter cells are placed in the logic hierarchy:

- `self` – The level-shifter cell is placed inside the model/cell being shifted.
- `parent` – The level-shifter cell is placed in the parent of the cell or model being shifted.
- `automatic` – The implementation tool is free to choose the appropriate location. This is the default behavior.

The `-applies_to_boundary` option specifies which power domain boundary to apply the strategy. For details, see Lower-Domain Boundary.

The `-name_prefix` and `-name_suffix` options let you control the naming of the level shifter cell instances inserted into the design that carry out the level shifter strategy.

The `-update` option allows you to provide additional information for a previous command with the same *`strategy_name`* and *`domain_name`* that was executed in the same scope. You must specify the `-update` option with either the `-elements` or `-exclude_elements` options. The resulting set of elements is the union of all elements specified in the element lists of all instances of the command.

The `-source` and `-sink` options allow you to refine the level-shifter strategy based on the supply powering the driver and load.

Specifying a strategy does not force a level shifter to be inserted unconditionally. The synthesis and physical implementation tools rely on the power state table and the specified rules (such as threshold) to determine where level shifters are needed. The tool issues a warning if it determines that a level shifter is not required.

The following strategies have decreasing level of precedence, irrespective of the order in which they are executed:

```
set_level_shifter -domain domain_name -elements ... [-applies_to ...]
set_level_shifter -domain domain_name -applies_to [inputs | outputs]
set_level_shifter -domain domain_name [-applies_to both]
```

It is an error to specify a strategy of the same precedence level explicitly on the same domain or design elements as a previous strategy specification.

By default, the synthesis and physical implementation tools can insert level shifters at domain boundaries even where no level shifter strategy has been defined. This gives the tool the flexibility to use whatever strategy is needed to get the best possible results. However, you can optionally restrict the tool to insert level shifters only at domain boundaries where the `set_level_shifter` command has been used to specify a strategy. To enforce this restriction, set the `upf_levshi_on_constraint_only` variable to `true`.

The synthesis and physical implementation tools ensure that level shifter insertion does not alter the logical function of the design. The generated gate-level netlist, with the exception of supply net connections for level shifters, is logically equivalent to the input

RTL. For this reason, Formality and VCS do not need to consider `set_level_shifter` commands.

map_level_shifter_cell

```
map_level_shifter_cell strategy_name
  -domain power_domain_name
  -lib_cells list
```

The `map_level_shifter_cell` command maps a particular level-shifter strategy to a library cell or range of library cells. All level-shifter cells belonging to the specified strategy are mapped to one of the library cells specified in the `-lib_cells` list. If a valid cell cannot be found in the list, no level-shifter cell is inserted.

The following example maps the level-shifter strategy called shift_up in the power domain PwrDomZ to the library cells LS_LH and LS_HL in library2.

```
map_level_shifter_cell shift_up -domain PwrDomZ \
  -lib_cells {/library2/LS_LH /library2/LS_HL}
```

use_interface_cell

```
use_interface_cell
  interface_implementation_name
  -domain domain_name
  -lib_cells lib_cell_list
  -strategy list_of_one_level_shifter_and_or_one_isolation
  [-port_map port_net_list]
  [-force_function]
```

The `use_interface_cell` command specifies the library cells to use for implementation of isolation and level-shifter strategies. This command replaces the deprecated `map_level_shifter_cell` command. If you specify both the `map_level_shifter_cell` and the `use_interface_cell` commands for the same strategy, the `use_interface_cell` has higher priority.

The `-port_map` and `-force_function` options are used by the simulation tools. They are read and ignored by the implementation tools.

name_format

```
name_format
  [-isolation_prefix string]
  [-isolation_suffix string]
  [-level_shift_prefix string]
  [-level_shift_suffix string]
```

When the synthesis or physical implementation tool inserts an isolation cell or level shifter into the design, it assigns an instance name to the new cell by adding a prefix to the beginning or a suffix to the end of the name of the object being isolated or level-shifted

(a port, pin, or net). The `name_format` command lets you specify the prefix or suffix to be used.

An empty string is a valid value for any prefix or suffix option.

Isolation Commands

The isolation commands specify the strategies for inserting isolation cells between the outputs of an “off” domain and the inputs of an “on” domain.

set_isolation

```
set_isolation isolation_strategy_name
  -domain power_domain
  [-elements objects]
  [-exclude_elements exclude_list]
  [-applies_to inputs | outputs | both]
  [-applies_to_boundary upper | lower | both]
  [-clamp_value  0 | 1 | latch]
  [-isolation_power_net isolation_power_net]
  [-isolation_ground_net isolation_ground_net]
  [-isolation_supply isolation_supply_set]
  [-source source_supply_set_name]
  [-sink sink_supply_set_name]
  [-diff_supply_only true | false]
  [-no_isolation]
  [-force_isolation]
  [-name_prefix prefix_string]
  [-name_suffix suffix_string]
  [-update]
```

The `set_isolation` command specifies an isolation strategy for a power domain. The strategy specifies the elements of the domain to which the strategy applies, the supply set or supply nets to be used for the isolation cells, and any insertion constraints based on the supply sets of the driver and receiver of the net being isolated. To complete the isolation strategy, you also need to use the `set_isolation_control` command to specify the isolation control signal, and optionally, the hierarchical location for inserting the cells on the domain boundary.

Synthesis and implementation tools insert isolation cells strictly according the isolation strategies specified by the `set_isolation` and `set_isolation_control` commands, without considering other information such as the power state table. If the isolation strategies are not consistent with the allowed states defined in the power state table, you can find and report these inconsistencies by using the `check_mv_design` command after the isolation cells are inserted.

Feedback

Where the Strategy Applies

By default, the strategy of a `set_isolation` command applies to all outputs that cross the boundary of the specified power domain.

To restrict the strategy to specific elements of the power domain, use the `-elements` option. The specified elements can be ports of the power domain or the pins of root cells on the boundary of the power domain. You can use the `set_isolation` command multiple times to create different isolation strategies for different elements in the power domain. You can also explicitly identify a set of ports to which the strategy does not apply by using the `-exclude_elements` option. If you use the `-elements` and `-exclude_elements` options with the `-update` option to add information to previous `set_isolation` commands issued in the same design scope, the set of instances or ports is the union of all elements specified with these options.

Both the `-elements` and `-exclude_elements` options accept the `{.}` as a valid element list. The `{.}` specifies all elements in the current scope.

The ports of a power domain are the logical ports of the root cells of the power domain, or in the case of a power domain containing the top-level design, the logical ports of the design. Input ports of a power domain are the ports defined as inputs in the corresponding HDL module. Similarly, output ports of a power domain are the ports defined as outputs in the corresponding HDL module.

The `-applies_to` option lets you specify explicitly whether to apply the strategy to inputs only, to outputs only, or to both inputs and outputs of the power domain:

- Applying the strategy to the outputs of a shutdown domain ensures that during shutdown, the outputs continue to drive the inputs of other domains with a known value (optionally specified by the `-clamp_value` option).
- Applying the strategy to the inputs of a power domain might be necessary if another domain drives the inputs, the other domain can be shut down, and the outputs of the other domain are not already isolated according to that domain's isolation strategy.

Without the `-applies_to` option, the strategy applies to both inputs and outputs of the domain if you use one or both of the `-source` and `-sink` options, or if you set the `-diff_supply_only` option to `true` and the supplies are defined by supply sets (rather than supply nets). Otherwise, the strategy applies to outputs only.

The `-applies_to_boundary` option allows you to specify which power domain boundary the strategy applies. This option allows you more flexibility on where to place the isolation cells. For details, see Lower-Domain Boundary.

The tool, by strictly following the isolation strategies, might insert more isolation cells than necessary, for example, at both the output of the driver domain and the input of the receiver domain. You can find such occurrences with the `check_mv_design` command. The tool merges redundant cells during optimization.

Feedback

Isolation Constraints Based on Source and Sink Supply Sets

The `-source`, `-sink`, and `-diff_supply_only` options restrict the insertion of isolation cells to only the domain-crossing nets whose driver and receiver cells meet specified supply set conditions. These options act as “filters” to reduce the number of ports on the domain boundary considered for isolation.

- The `-source` option specifies a supply set name. The isolation strategy applies only to the domain-crossing nets that are driven by a cell powered by the specified supply set.
- The `-sink` option specifies a supply set name. The isolation strategy applies only to the domain-crossing nets that fan out to cells powered by the specified supply set.

  You can use both the `-source` and `-sink` options to restrict the insertion of isolation cells to only the domain-crossing nets that are driven by a cell powered by a specified supply set and that fan out to cells powered by a different specified supply set.
- The `-diff_supply_only` option, when set to `true`, applies the strategy only to the domain-crossing nets in which the driver and receivers are powered by different supply sets or supply nets. This setting prevents isolation when the driver and receiver cells use matching supplies. The default is `false`, meaning no restriction on isolation based on matching driver and receiver supplies.

Setting the `-diff_supply_only` option to `true` restricts the isolation strategy to all combinations of differing source and sink supply sets, whereas setting the `-source` and `-sink` options restrict the strategy to specifically named combinations of supply sets. Note that you cannot use the `-diff_supply_only`, `-source`, and `-sink` options all together in the same strategy.

Isolation Cell Supply Set or Supply Nets

The power and ground supplies for the inserted isolation cells must be active while the driver domain is shut down and the receiver domain is active. For each isolation strategy, the power supplies for the isolation cells must be specified by one of the following options:

- The `-isolation_supply` option of the `set_isolation` command
- The `-isolation_power_net` and `-isolation_ground_net` options of the `set_isolation` command
- The `-supply {default_isolation` *`supply_set_name`*`}` option of the `create_power_domain` command

Using the `-isolation_supply` option of the `set_isolation` command completely specifies the supplies for the isolation cells. Alternatively, you can specify the power and ground supply nets explicitly using one or both of the `-isolation_power_net` and `-isolation_ground_net` options. If you specify only the power net or only the ground net, the primary supply of the power domain is used for the unspecified supply net.

Feedback

If you specify `-isolation_supply {}` (using an empty set as the supply set argument) together with the `-clamp_value 0` option, the tool uses a single-rail, clamp-to-zero isolation cell that lacks an isolation supply rail. This type of isolation cell can be implemented efficiently as a NOR gate with an inverter on the data input.

If you use none of these options in the `set_isolation` command, the `default_isolation` supply set specified by the `-supply` option of the `create_power_domain` command applies.

Clamp Value

The `-clamp_value` option specifies the constant value of the isolation cell output when the isolation function is enabled. It can be set to `0`, `1`, or `latch`. A clamp value of `0` (the default) can be implemented by an AND gate. A clamp value of `1` can be implemented by an OR gate. A clamp value of `latch` is implemented by an isolation cell that latches the current data, either 0 or 1, when the isolation function is enabled, and holds that value constant during isolation.

Forcing or Preventing Isolation

In case of conflicting isolation strategies, the tool follows certain precedence rules to determine which strategy to use. For details, see the *Power Compiler User Guide*.

Using the `-force_isolation` option of the `set_isolation` command ensures that an isolation strategy is applied to the specified elements, irrespective of the default priority order. The isolation cells inserted with this option cannot be optimized away by implementation tools.

To prevent the insertion of isolation cells for the specified elements of the power domain, use the `-no_isolation` option. In that case, there is no need to specify the isolation power supplies for the strategy, and there is no need for the `set_isolation_control` command for the strategy.

Isolation Cell Naming Conventions

The `-name_prefix` and `-name_suffix` options let you control the naming of the isolation cell instances inserted into the design that carry out the isolation strategy, overriding the settings specified by the `name_format` command.

set_isolation_control

```
set_isolation_control isolation_strategy_name
  -domain power_domain
  -isolation_signal isolation_signal
  [-isolation_sense  high | low]
  [-location self | parent | fanout]
```

The `set_isolation_control` command specifies the isolation control signal for a specified isolation strategy, and optionally, the hierarchical location for inserting isolation cells on the domain boundary.

The `-isolation_signal` argument specifies the name of the control signal that enables and disables the isolation function of cells created by the isolation strategy. You can specify control signal as a port, pin, or net. A port or pin has priority over an identically named net.

The `-isolation_sense` option specifies the logic state of the isolation control signal, either `high` or `low`, that enables the isolation function of the isolation cells. The default is `high`.

The isolation signal need not exist in the logical hierarchy where the isolation cells are to be inserted; the synthesis or implementation tool can perform port-punching as needed to make the connection. Port-punching means automatically creating a port to make a connection from one hierarchical level to the next.

Punched ports are not considered for isolation or level shifting, even though these ports might reside within the coverage of an isolation or level shifter strategy. Existing ports, even if they reside on an always-on path, are isolated and level-shifted according to the applicable isolation and level shifter strategy.

The `-location` option setting specifies where to insert isolation cells with respect to the power domain boundary:

- `self` (the default) places isolation cells in the design being isolated, just inside the power domain boundary.
- `parent` places isolation cells in the parent of the design being isolated, just outside the power domain boundary.
- `fanout` places the isolation cells at all fanout locations (sinks) of the port being isolated. This option is valid only for an isolation strategy defined by the `set_isolation` command using the `-source` and `-sink` options or the `-diff_supply_only true` setting.

Using the `fanout` setting might be necessary when an output port fans out to multiple power domains, each requiring a different type of isolation.

For more information about the rules that apply when you use the `-location fanout` option, see the *Power Compiler User Guide*.

map_isolation_cell

```
map_isolation_cell isolation_strategy_name
  -domain power_domain_name
  [-lib_cells list]
```

The `map_isolation_cell` command maps a particular isolation strategy to a library cell or range of library cells. All isolation cells belonging to the specified strategy are mapped or remapped to one of the library cells specified in the `-lib_cells` list.

The following example maps the isolation strategy called test_PD1 in the power domain PD1 to the library cells ISO_L and ISO_H in library2.

```
prompt> map_isolation_cell test_PD1 -domain PD1 \
  -lib_cells {/library2/ISO_L /library2/ISO_H}
```

use_interface_cell

```
use_interface_cell
  interface_implementation_name
  -domain domain_name
  -lib_cells lib_cell_list
  -strategy list_of_one_level_shifter_and_or_one_isolation
  [-port_map port_net_list]
  [-force_function]
```

The `use_interface_cell` command specifies the library cells to use for implementation of isolation and level-shifter strategies. This command replaces the deprecated `map_isolation_cell` command. If you specify both the `map_isolation_cell` and the `use_interface_cell` commands for the same strategy, the `use_interface_cell` has higher priority.

The `-port_map` and `-force_function` options are used by the simulation tools. They are read and ignored by the implementation tools.

name_format

```
name_format
  [-isolation_prefix string]
  [-isolation_suffix string]
  [-level_shift_prefix string]
  [-level_shift_suffix string]
```

When a synthesis or physical implementation tool inserts an isolation cell or level shifter into the design, it assigns an instance name to the new cell by adding a prefix to the beginning or a suffix to the end of the name of the object being isolated or level-shifted (a port, pin, or net). The `name_format` command lets you specify the prefix or suffix to be used.

An empty string is a valid value for any prefix or suffix option.

Retention Commands

The retention commands specify the strategy for inserting retention cells inside switched (power-down) domains.

Feedback

set_retention

```
set_retention retention_strategy_name
  -domain power_domain
  [-retention_power_net retention_power_net]
  [-retention_ground_net retention_ground_net]
  [-retention_supply retention_supply_set]
  [-no_retention]
  [-elements objects]
  [-exclude_elements objects]
  [-save_condition boolean_function]
  [-restore_condition boolean_function]
  [-retention_condition boolean_function]
  [-use_retention_as_primary]
  [-update]
```

The `set_retention` command specifies which registers in the domain are to be implemented as retention registers and identifies the save and restore signals for the retention functionality. Only the registers implied in the elements are given retention capabilities.

The `-elements` option specifies the objects in the power domain to which the retention strategy applies. The objects can be hierarchical cells, leaf-level cells, HDL blocks, and nets. If a design element is specified, then all registers within the design element acquire the specified retention strategy. If a process is specified, then all registers inferred by the process acquire the specified retention strategy. If a register, signal, or variable is specified and that object is a sequential element, then the implied register acquires the specified retention strategy. Any specified register, signal, or variable that does not infer a sequential element is not affected by this command. If `-elements` is not used, it is the same as listing the elements that define the power domain.

If the `-elements` option is not used, the retention strategy is applied to elements inferred by the strategy, unless the `-no_retention` option is used.

Both the `-elements` and `-exclude_elements` options accept the `{.}` as a valid element list. The `{.}` specifies all elements in the current scope. For following example, the ret strategy is only applied to the elements in the TOP domain. Elements in B1/C1 are not included.

```
set_scope
create_power_domain TOP -elements {. B1/C1}
set_retention ret -domain TOP -elements {.}
```

At a minimum, either `-retention_power_net` or `-retention_ground_net` must be used to specify a retention register. If both are used, they specify the supply nets to use as the retention power and ground nets. If only the retention power supply net is specified, the primary ground net is used as the retention ground supply. If only the retention ground net is specified, the primary supply net is used as the retention power supply. The retention

power and ground nets are automatically connected to the implicit save and restore processes and the shadow register.

The `-retention_supply` option specifies that the power and ground functions of the supply set are used as the retention power and retention ground nets, respectively. The `-retention_supply_set` is mutually exclusive with the `-retention_power_net` and `-retention_ground_net` options.

The `-no_retention` option prevents retention behavior for the objects specified in the command, allowing you to specify exceptions to a more broadly applied retention strategy.

The `-save_condition` option specifies a Boolean expression that controls the saving of the value in the register into the state-saving latch of the register.

The `-restore_condition` specifies the Boolean condition to restore the value from the state-saving latch into the register.

The `-retention_condition` option specifies the Boolean condition to preserve the value saved in the state-saving register.

The `-update` option allows you to refine the element list of a previously defined retention strategy. When used with the `-elements` option, the set of elements is the union of all elements specified for a strategy. You cannot redefine a domain-based retention strategy to an element-based retention strategy with the `-update` option.

The following strategies have decreasing level of precedence, irrespective of the order in which they are executed:

```
set_retention -domain -elements
set_retention -domain
```

The isolation power and ground nets should not operate at the same voltages as the primary power and ground nets of the power domain where the retention cells will be located.

Every retention strategy defined without the `-no_retention` option must have a corresponding `set_retention_control` command.

set_retention_control

```
set_retention_control retention_strategy_name
  -domain power_domain
  -save_signal {save_signal high | low}
  -restore_signal {restore_signal high | low}
  [-assert_r_mutex net_name high | low]*
  [-assert_s_mutex net_name high | low]*
  [-assert_rs_mutex net_name high | low]*
```

*Asterisk indicates that the item in square braces can be repeated in the command.

Feedback

The `set_retention_control` command allows the specification of the retention control signal and the logical sense of that signal. The command identifies an existing retention strategy and specifies the save and restore signals and logical senses of those signals for that strategy.

The `-save_signal` setting specifies the existing net, port, or pin in the design used to save data into the shadow register before power-down; and the logic state of the signal, either low or high, that causes this action to be taken.

Similarly, the `-restore_signal` setting specifies the existing net, port, or pin in the design used to restore data from the shadow register before power-up; and the logic state of the signal, either low or high, that causes this action to be taken.

Each control signal can be either a net or a pin/port, with net having higher precedence. The retention signal need not exist in the logical hierarchy where the retention cells are be inserted; the synthesis or implementation tool can perform port-punching as needed to make the connection. Port-punching means automatically creating a port to make a connection from one hierarchical level to the next. These punched ports are not considered for isolation or level-shifting, even though after the port creation, these ports reside within the coverage of an isolation or level-shifter strategy.

Existing ports, even if they reside on an always-on path (a logic path marked as always-on relative to the receiving end), are isolated and level-shifted according to the applicable isolation and level-shifter strategy.

The `-assert_r_mutex, -assert_s_mutex, and -assert_rs_mutex` options create assertions. These assertions are used as triggers by the verification and simulation tools. These assertions are triggered when the signal on the specified net and the save signal, restore signal, or both specified by the `-save_signal`, `-restore_signal` option, or both are active simultaneously. You can specify these options multiple times.

Note:

The `-assert_r_mutex`, `-assert_s_mutex`, and `-assert_rs_mutex` options are parsed by the Design Compiler, IC Compiler, IC Compiler II, and Fusion Compiler tools, but not used during implementation. Also, the `save_upf` command does not write these options.

set_retention_elements

```
set_retention_elements retention_list_name
   -elements element_list
```

The `set_retention_elements` command creates a list of elements whose collective state is maintained if the retention is applied to any of these elements. The list of elements applies to the scope where the command is defined. If all of the elements are not retained, while any one element is in a retention state, it is an error. You can also use the `retention_list_name` in subsequent `set_retention` commands.

Feedback

map_retention_cell

```
map_retention_cell retention_strategy_name
  -domain power_domain
  [-lib_cells lib_cells]
  [-lib_cell_type lib_cell_type]
  [-lib_model_name model_name {-port port_name net_ref}*]
  [-elements objects]
```

*Asterisk indicates that the item in curly braces can be repeated in the command.

The `map_retention_cell` command provides a mechanism for constraining the implementation choices for retention registers. The command must specify the name of an existing retention strategy and power domain.

The `-lib_cells` option specifies a list of target library cells to be used for retention mapping.

The `-lib_cell_type` option directs the implementation tool to select a retention cell that has the specified cell type, according to the `lib_cell_type` attribute. Note that this option setting does not change the simulation semantics specified by the `set_retention` command.

The `-lib_model_name` option specifies the name of the verification model and associated port connectivity. This information is used by verification tools.

The `-elements` option lists the register elements (directly or indirectly) within the domain to which the mapping command is applied. The elements must be included in the elements listed in the related `set_retention` command. If `-elements` is not used, all registers inferred from the retention strategy have the mapping applied.

The `map_retention_cell` command accepts both UPF version 2.0 and 2.1 syntax for the `-lib_model_name` option. For example, the tool accepts both of the following versions:

```
map_retention_cell s1_pd -domain core1_pd \
   ... \
   -lib_model_name MODEL1 { \
      -port SAVE save_signal \
      -port RESTORE restore_signal \
      -port VDD core1_pd.primary.power \
      ...}
map_retention_cell s1_pd -domain core1_pd \
   ... \
   -lib_model_name MODEL1 \
   -port_map { \
       {SAVE save_signal} \
       {RESTORE restore_signal} \
       {VDD core1_pd.primary.power \
        ...}
```

Feedback

Power Model Commands

When you use macros, there are often many instances of the same macro in your design. Typically, each macro instance has the same UPF loaded in its scope, which causes the tool to call the `load_upf` command each time the macro is used. To minimize the use of the `load_upf` command, you can use a power model to define the UPF for the macro.

To use a power model, first define a power model using the `define_power_model` command. When you define a power model, specify the list of UPF commands that define the power model architecture. The `define_power_model` command provides a protected environment for the power model. Any Tcl variable used in the `define_power_model` command is not affected by the global variables defined outside the model definition. To specify any parameters for the power model, use the `add_parameter` command.

After you define a power model and specify any parameters, you can instantiate it using the `apply_power_model` command. To override any parameter settings made in the model, use the `-parameters` option of the `apply_power_model` command.

You can use power models in the hierarchical flow. The `split_constraints` and `propagate_constraints` commands support power models. For the `split_constraints` command, the `define_power_model` and `apply_power_model` commands and any corresponding attributes are included in the top-level design and in blocks where the power model is applied. When propagating UPF constraints from linked blocks, the tool applies models based on the power model commands and attributes from the block UPF.

Note:

The Design Compiler tool does not support power models in the hierarchical flow.

add_parameter

```
add_parameter
  parameter_name
  -default default_value
  [-type type_name]
  [-description description_text]
```

The `add_parameter` command is used to define parameters to use within a power model. The *`parameter_name`* specifies the name of the parameter. When the `-type` option is specified, the parameter defined is used for system-level IP power models. The type can be one of the following: `buildtime`, `runtime`, or `rate`. The scope of a power model parameter resides solely within the model in which it is defined.

apply_power_model

```
apply_power_model
  [power_model_name]
  [-elements instance_names]
```

```
  [-supply_map list_of_supply_maps]
  [-port_map list_of_port_maps]
  [-parameters list_of_parameters]
  [-all_hard_macros]
```

The `apply_power_model` command binds the *`power_model_name`* to the cell instances specified by the `-elements` option. The `-parameters` option sets the values of parameters previously defined by the `add_parameter` command for the instances listed in the `-elements` option. You an only set the values of existing parameters in the model, not create new ones. The `-parameter` option overrides any parameters defined using the `add_parameter` command.

The `-port_map` option maps supply ports or logic ports in the power model to supply nets or logic nets in the current scope. The `-supply_map` option maps supply sets in the power model to supply sets in the current scope.

The `-all_hard_macros` option specifies that power models are applied to all hard macro cells under the current scope.

define_power_model

```
define_power_model
  power_model_name
  [-for list_of_library_cells]
  {UPF_commands}
```

The `define_power_model` command defines a power model that contains UPF commands within the curly braces. These UPF commands specify the power intent of the power model. The `apply_power_model` command loads these UPF commands into the scope of the target cells.

The `-for` option specifies a collection of library cells to which this power model applies.

You can reference the power model by its simple name from anywhere in the power intent description. The syntax of the UPF commands is not checked until the `apply_power_model` command is run.

Power State Table Commands

A power state table defines the legal combinations of states that can exist at the same time during operation of a design. The state of a power domain consists of the set of on-off states and voltage levels of its supply ports. The power state table is used for synthesis, analysis, and optimization.

create_pst

```
create_pst table_name
  -supplies list
```

The `create_pst` command creates a new power state table and assigns a name to the table. The command lists the supply ports or supply nets in a particular order. The `add_port_state` command defines the names of the possible states for each supply port. The `add_pst_state` command lists the allowed combinations of states in the design.

The `create_pst` command can only be used at the top-level scope. Power switch supply ports are considered supply ports because they are connected by supply nets, so they can be listed as supply nets in the `create_pst` command.

A supply port and a supply net can have the same name, even when they are unconnected. If such a name is listed in the `create_pst` command, it is assumed to represent the supply port and not the supply net.

add_port_state

```
add_port_state port_name
  {-state {name nom | min nom max | off}}*
```

*Asterisk indicates that the item in curly braces can be repeated in the command.

The `add_port_state` command adds state information to a supply port. The command specifies the name of the supply port and the possible states of the port. Each state is specified as a state name and the voltage level for that state. The voltage level can be specified as a single nominal value, a set of three values (minimum, nominal, and maximum), or the keyword `off` to indicate the “off” state. The first state specified is the default state of the supply port.

You can specify a nominal voltage or a range of voltages for a state. However, a port might be in any of the defined states and can take any voltage within a defined range, so a tool cannot determine the voltage to use for optimizing or analyzing the design from the power state table alone. In the Design Compiler, IC Compiler, IC Compiler II, Fusion Compiler, and PrimeTime tools, you use the `set_voltage` command (a non-UPF command) to specify the operating voltage for optimization or analysis.

Voltages and port states are not propagated through power switches. Therefore, you need to define port states for the output ports of power switches and add those states to the power state table.

The `add_port_state` command can be used to represent off-chip supply sources that are not driven by the testbench. The specification of the supply port voltage provides a convenient shortcut to facilitate verification and analysis without requiring the creation of a power domain and a supply network within the verification environment.

A power switch supply port is considered a supply port because it is connected by a supply net, so it can be specified as the supply port in the `add_port_state` command. Similarly, an RTL port can be made a supply port by this command or by the `connect_supply_net` command. Note that supply states specified at different supply ports

are shared within a group of supply nets and supply ports directly connected together. However, this sharing does not happen across a power switch.

add_pst_state

```
add_pst_state state_name
  -pst table_name
  -state list_of_supply_states
```

The `add_pst_state` command defines the states of each of the supply nets for one possible state of the design. The command must specify a name for the state, the name of the power state table previously created by the `create_pst` command, and the states of the supply ports in the same order listed in the `create_pst` command.

The listed states must match the supply ports or nets listed in the `create_pst` command in corresponding order. For a group of supply ports and supply nets directly connected together, the allowable supply states are derived from the shared pool of supply states commonly owned by the members of the group.

The following example creates a power state table, defines the states for the supply ports, and lists the allowed power states for the design.

```
create_pst pt -supplies { PN1 PN2 SOC/OTC/PN3 }
add_port_state PN1 -state {s88 0.88}
add_port_state PN2 -state {s88 0.88} -state {s99 0.99}
add_port_state SOC/OTC/PN3 -state {s88 0.88} -state {pdown off}
add_pst_state s1 -pst pt -state { s88 s88 s88 }
add_pst_state s2 -pst pt -state { s88 s88 pdown }
add_pst_state s3 -pst pt -state { s88 s99 pdown }
```

You can use multiple power state tables to specify the states within a particular scope. In the case of independent power supplies, those supplies can be specified as smaller, independent power state tables. The tool expands the specified tables to cover all possible combinations of states.

For example, suppose you have power supplies VDD1 and VDD1sw, which operate independently from power supplies VDD2 and VDD2sw. The “sw” supplies can be switched on and off independently. You can specify the power states as follows:

```
create_pst table1 -supplies { VDD1 VDD1sw }
add_port_state VDD1 -state {HV 1.2}
add_port_state VDD1sw -state {HV 1.2} -state {OFF 0.0}
add_pst_state s1 -pst table1 -state { HV HV }
add_pst_state s2 -pst table1 -state { HV OFF }
create_pst table2 -supplies { VDD2 VDD2sw }
add_port_state VDD2 -state {HV 1.2}
add_port_state VDD2sw -state {HV 1.2} -state {OFF 0.0}
add_pst_state s1 -pst table2 -state { HV HV }
add_pst_state s2 -pst table2 -state { HV OFF }
```

Feedback

The tool internally expands these two tables, producing the same combinations of power states as the following single table:

```
create_pst table -supplies { VDD1 VDD1sw VDD2 VDD2sw}
add_port_state VDD1 -state {HV 1.2}
add_port_state VDD1sw -state {HV 1.2} -state {OFF 0.0}
add_port_state VDD2 -state {HV 1.2}
add_port_state VDD2sw -state {HV 1.2} -state {OFF 0.0}
add_pst_state s1 -pst table -state { HV  HV HV  HV }
add_pst_state s2 -pst table -state { HV  HV HV OFF }
add_pst_state s3 -pst table -state { HV OFF HV  HV }
add_pst_state s4 -pst table -state { HV OFF HV OFF }
```

If the scope has a large number of power supplies, specifying the states using multiple tables can be more compact and simpler than using a single large table.

The wildcard character, an asterisk, can be used to indicate that all the states of the corresponding port are valid for that table state. For example,

```
add_pst_state s0 -pst table -state { HV * HV * }
```

To implement footer switches in switched power domains, you need to define the states of the applicable ground port. The `create_pst` and `add_pst_state` commands allow ground nets to be included in their states, and the `add_port_state` command accepts both 0.00 and `off` as valid states for a ground port.

add_power_state

```
add_power_state
  [-domain | -supply | -group]
  object_name
  -state state_name
    { [-supply_expr {supply_expression}
      [-logic_expr {logic_expression}]
      [-simstate {simstate}]
      [-illegal] }*
  [-simstate {simstate}]
  [-update]
```

*Asterisk indicates that the item in curly braces can be repeated in the command.

The `add_power_state` command does one of the following.

- Sets the power states on the nets of a *domain*:

  `add_power_state` **`-domain`** ***`domain_name`*** `-state state_name ...`

  This creates a power state for the specified domain.

- Sets the power states on the nets of a *supply set*:

  `add_power_state` **`-supply`** ***`supply_set_name`*** `-state` *`state_name`* `...`

This allows the states to be used in a power state table.

- Sets the power states of a *power state group*:

  ```
  add_power_state -group group_name -state state_name ...
  ```

  This creates a power state table for the design. You create power state groups by using the `create_power_state_group` command.

- If none of the options, `-domain`, `-group`, or `-supply`, are specified, then the *`object_name`* type determines the kind of object to which the command applies.

Adding Power States to a Supply Set

Use the `add_power_state` command with the `-supply` option to add power states to a supply set. This allows the defined states to be used in the power state table. The command specifies the possible supply states by supply set and by function (power or ground).

For example,

```
add_power_state -supply SS1
  -state my_ON \
    {-supply_expr {power == {FULL_ON 0.8} && ground == {FULL_ON 0.0}}}
add_power_state -supply SS2
  -state my_ON \
    {-supply_expr {power == {FULL_ON 0.8} && ground == {FULL_ON 0.0}}}
  -state my_OFF \
    {-supply_expr {power == {OFF} && ground == {FULL_ON 0.0}}}
```

This example creates the state `my_ON` for supply set `SS1` and states `my_ON` and `my_OFF` for supply set `SS2`. For each state and for each supply function, the command specifies the state (either `FULL_ON` or `OFF`) and the voltage of the supply net (if `FULL_ON`).

In the `add_power_state` command:

- The `-supply` option specifies the name of the supply set. The `-supply` keyword is optional because the default action is to add states to the specified supply set.
- The `-state` option specifies a name for the power state being created for the supply set.
- The `-supply_expr {`*`supply_expression`*`}` option specifies a power net (either `power` or `ground`), the state of that net (either `FULL_ON` or `OFF`), and the voltage of that net.

  In general, the *`supply_expression`* argument has the following form:

  ```
  power == netstate [ && ground == netstate ]
  ```

where *netstate* has one of the following forms:

```
   status
{status}
{status nom}
{status min max}
{status min nom max}
```

where *status* is either FULL_ON or OFF and *nom*, *min*, and *max* are the nominal, minimum, and maximum voltage values.

The following older syntax is also supported:

```
`{status}
`{status, nom}
`{status, min, max}
`{status, min, nom, max}
```

However, you should use the newer syntax in all new add_power_state commands.

- The -logic_expr option specifies a SystemVerilog Boolean expression in terms of logic nets, supply nets, and logic signals.
- The -simstate option specifies a simulation state for the power states associated with a supply set. It is used only by the VCS NLP tool. The supported values are NORMAL, CORRUPT, and CORRUPT_ON_ACTIVITY.
- The -illegal option specifies that the defined state is not allowed. Simulation tools check for the design entering forbidden states. This option setting is not used by synthesis or implementation tools.
- The -update option specifies that the state is to be added to the existing defined states of the supply set, without replacing the existing states.

  You can also use the -update option to specify voltages that were previously unspecified. For instance,

```
# an unspecified UPF voltage
add_power_state SS1 -state ON {-supply_expr {power =={FULL_ON}}}
...
# specify the UPF voltage with -update
add_power_state SS1 -state ON \
  {-supply_expr {power == {FULL_ON 0.8}}} -update
```

Adding States to a Power State Group

Use the add_power_state command with the -group option to add power states to a power state group previously created with the create_power_state_group command. This creates a power state table without using create_pst or add_pst_state commands.

Feedback

For example,

```
add_power_state -supply SS1
  -state my_ON \
    {-supply_expr {power == {FULL_ON 0.8} && ground == {FULL_ON 0}}}

add_power_state -supply SS2
  -state my_ON \
    {-supply_expr {power == {FULL_ON 0.8} && ground == {FULL_ON 0.0}}}
  -state my_OFF \
    {-supply_expr {power == {OFF} && ground == {FULL_ON 0.0}}}

create_power_state_group BLK1_PST

add_power_state -group BLK1_PST \
   -state RUN   {-logic_expr {SS1 == my_ON && SS2 == my_ON}} \
   -state SLEEP {-logic_expr {SS1 == my_ON && SS2 == my_OFF}}
```

The final `add_power_state` command in this example creates two power states, `RUN` and `SLEEP`, in group `BLK1_PST`, and defines the supply set states corresponding to each group state. This creates the following power state table.

Figure 26 Power State Table Created by the add_power_state Command

State	SS1	SS2
RUN	ON (0.8V)	ON (0.8V)
SLEEP	ON (0.8V)	OFF

In the `add_power_state` command:

- The `-group` option specifies the name of the power state group previously created with the `create_power_state_group` command.
- The `-state` option specifies a name for the state being created for the group.
- The `-logic_expr` option specifies a Boolean expression in terms of supply sets, other power state groups, or existing power state tables and their states. Only the `&&` operator is allowed in the expression. If a logic signal is specified, it can only be used with binary values of 0 or 1.
- The `-illegal` option specifies that the defined state is not allowed. Simulation tools check for the design entering forbidden states. This option setting is not used by synthesis or implementation tools.
- The -`update` option specifies that the state is to be added to the existing defined states of the group, without replacing the existing states.

The `-supply_expr` and `-simstate` options cannot be used with the `-group` option.

Composite Power State Groups

You can use multiple power state groups to concisely specify combinations of allowed states in the design. For example, you have six supply sets, SS1 through SS6. You divide them into two groups of related supply sets, P1 (SS1, SS2, SS3) and P2 (SS4, SS5, SS6), and define states for these groups as follows:

```
add_power_state -supply SS1 \
  -state my_ON \
    {-supply_expr {power == {FULL_ON 0.8} && ground == {FULL_ON 0}}}

add_power_state -supply SS2 \
  -state my_ON \
    {-supply_expr {power == {FULL_ON 0.8} && ground == {FULL_ON 0.0}}}
  -state my_OFF \
    {-supply_expr {power == {OFF} && ground == {FULL_ON 0.0}}}

...

create_power_state_group P1
add_power_state -group P1 \
   -state P1_RUN   \
      {-logic_expr {SS1 == my_ON && SS2 == my_ON  && S3 == my_ON}}  \
   -state P1_IDLE  \
      {-logic_expr {SS1 == my_ON && SS2 == my_ON  && S3 == my_OFF}} \
   -state P1_SLEEP \
      {-logic_expr {SS1 == my_ON && SS2 == my_OFF && S3 == my_OFF}}

create_power_state_group P2
add_power_state -group P2
   -state P1_RUN   \
      {-logic_expr {SS1 == my_ON &&  SS2 == my_ON  && S3 == my_ON}}  \
   -state P1_IDLE  \
      {-logic_expr {SS1 == my_ON &&  SS2 == my_ON  && S3 == my_OFF}} \
   -state P1_OFF   \
      {-logic_expr {SS1 == my_OFF && SS2 == my_OFF && S3 == my_OFF}}
```

These commands create the power state tables shown in the following figure.

Feedback

Figure 27 Power State Table Created by add_power_state Command

Group P1

State	SS1	SS2	SS3
P1_RUN	ON	ON	ON
P1_IDLE	ON	ON	OFF
P1_SLEEP	ON	OFF	OFF

Group P2

State	SS4	SS5	SS6
P2_RUN	ON	ON	ON
P2_IDLE	ON	ON	OFF
P2_OFF	OFF	OFF	OFF

At this point, you can create a higher-level power state group called PMAIN and define states for it by using the states of groups P1 and P2, as follows:

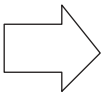
```
create_power_state_group PMAIN
add_power_state -group PMAIN \
   -state RUN   {-logic_expr {P1 == P1_RUN   &&  P2 == P2_RUN }} \
   -state INIT  {-logic_expr {P1 == P1_RUN   &&  P2 == P2_OFF }} \
   -state IDLE  {-logic_expr {P1 == P1_IDLE  &&  P2 == P2_RUN }} \
   -state SLEEP {-logic_expr {P1 == P1_IDLE  &&  P2 == P2_IDLE}} \
   -state DOWN  {-logic_expr {P1 == P1_SLEEP &&  P2 == P2_OFF }}
```

This creates the composite power state table shown in the following figure.

Figure 28 Composite Power State Table Using Group States

Group PMAIN

State	Group P1	Group P2
RUN	P1_RUN	P2_RUN
INIT	P1_RUN	P2_OFF
IDLE	P1_IDLE	P2_RUN
SLEEP	P1_IDLE	P2_IDLE
DOWN	P1_SLEEP	P2_OFF



Equivalent single-level PST

SS1	SS2	SS3	SS4	SS5	SS6
ON	ON	ON	ON	ON	ON
ON	ON	ON	OFF	OFF	OFF
ON	ON	OFF	ON	ON	ON
ON	ON	OFF	ON	ON	OFF
ON	OFF	OFF	OFF	OFF	OFF

In general, the logic expression for the state of a power state group can contain a Boolean expression in terms of supply sets, other power state groups, or existing power state tables created by the `create_pst` command.

Feedback

Hierarchical Power State Groups

You can compose a power state table hierarchically using different power state groups for multiple lower-level blocks. For example, you have two lower-level blocks, BLK1 and BLK2. You create power state groups at the scopes of these blocks:

```
add_power_state -supply SS1
  -state my_ON \
    {-supply_expr {power == {FULL_ON 0.8} && ground == {FULL_ON 0}}}

add_power_state -supply SS2
  -state my_ON \
    {-supply_expr {power == {FULL_ON 0.8} && ground == {FULL_ON 0.0}}}
  -state my_OFF \
    {-supply_expr {power == {OFF} && ground == {FULL_ON 0.0}}}

set_scope BLK1
create_power_state_group my_PST
add_power_state -group my_PST \
   -state RUN   {-logic_expr {SS1 == my_ON && SS2 == my_ON}} \
   -state SLEEP {-logic_expr {SS1 == my_ON && SS2 == my_OFF}}
set_scope ..

set_scope BLK2
create_power_state_group my_PST
add_power_state -group my_PST \
   -state RUN   {-logic_expr {SS1 == my_ON && SS2 == my_ON}} \
   -state SLEEP {-logic_expr {SS1 == my_ON && SS2 == my_OFF}}
set_scope ..
```

Now you can create a top-level power state group composed of lower-level groups:

```
create_power_state_group my_TOP_PST

add_power_state -group my_TOP_PST \
 -state RUN \
   {-logic_expr {BLK1/my_PST == RUN   && BLK2/my_PST == RUN   }} \
 -state SAVE \
   {-logic_expr {BLK1/my_PST == SLEEP && BLK2/my_PST == RUN   }} \
 -state SLEEP \
   {-logic_expr {BLK1/my_PST == SLEEP && BLK2/my_PST == SLEEP }}
```

This creates the following top-level power state table.

Feedback

Figure 29 Hierarchical Power State Table Created by the add_power_state Command

State	BLK1/SS1	BLK1/SS2	BLK2/SS1	BLK2/SS2
RUN	ON	ON	ON	ON
SAVE	ON	OFF	ON	ON
SLEEP	ON	OFF	ON	OFF

create_power_state_group

```
create_power_state_group group_name
```

The `create_power_state_group` command creates a new power state group and assigns a name to the group. You can then use the `add_power_state -group` command to add states to the group, thereby creating a power state table that defines the allowed power states of the design. For example,

```
create_power_state_group BLK1_PST

add_power_state -group BLK1_PST \
   -state RUN   {-logic_expr {SS1 == my_ON && SS2 == my_ON}} \
   -state SLEEP {-logic_expr {SS1 == my_ON && SS2 == my_OFF}}
```

For more information, see the description of the `add_power_state` command.

add_state_transition

```
add_state_transition
  [-supply | -domain | -group | -model | -instance] object_name
  -transition {transition_name [-from from_list -to to_list]
                               [-paired {from_state to_state}]
                               [-through through_list]
                               [-legal | illegal]}
  [-update]
```

The `add_state_transition` command defines named state transitions between power states of an object. This command replaces the `describe_state_transition` command in the IEEE 1801 standard, but Synopsys tools support both commands.

describe_state_transition

```
describe_state_transition
  transition_name
  -object object_name
  [-from from_list -to to_list]
  [-paired {from_state to_state}]
```

```
  [-through through_list]
  [-legal | -illegal]
```

The `describe_state_transition` command specifies the legality of a transition from one object's named power state to another power state.

Logic Editing Commands

The logic editing commands create logic nets and logic ports and make connections between these nets and ports, irrespective of the power network. These commands have an effect only in the Design Compiler, IC Compiler II, and Fusion Compiler tools. In other tools such as IC Compiler, PrimeTime, and Formality, the commands are accepted as valid syntax but are otherwise ignored.

create_logic_net

```
create_logic_net net_name
```

The `create_logic_net` command creates a new logic net in the active scope. This command has an effect only in synthesis tools.

create_logic_port

```
create_logic_port port_name
  [-direction in | out | inout]
```

The `create_logic_port` command creates a new logic port in the active scope and specifies the port direction (`in` for input, `out` for output, or `inout` for bidirectional). This command has an effect only in synthesis tools.

connect_logic_net

```
connect_logic_net net_name
  -ports port_list
```

The `connect_logic_net` command connects a specified logic net to one or more specified logic ports. This command has an effect only in synthesis tools.

Utility Commands

There are commands to load and execute UPF commands from a file, to write a set of UPF commands to a file, and to set the hierarchical scope for subsequent UPF commands.

load_upf

```
load_upf upf_file_name
  [-supplemental supf_file_name]
```

```
  [-scope string]
  [-noecho]
  [-simulation_only]
  [-strict_check string]
```

The `load_upf` command executes the UPF commands in a specified file. The commands describe the power intent of the design in the same way that the commands read by the `read_sdc` command describe the timing constraints on a design.

You can optionally specify a scope for the command. The scope is a name of a hierarchical instance on which the commands are applied. Specifying a scope has the same effect as changing the tool to the level of instance with the `set_scope` or `current_instance` command, executing the commands, and then changing back to the original hierarchical level. If you do not specify a scope, the commands are applied to the current scope.

The `-simulation_only` option causes synthesis and implementation tools to read but ignore the UPF command file and pass the commands on to downstream simulation tools. The Design Compiler tool lets you remove loaded UPF commands with the `remove_upf` command.

The `-supplemental` and `-strict_check` options are related to usage in the golden UPF mode. For details, see SolvNet article 1412864, “Golden UPF Flow Application Note.”

save_upf

```
save_upf upf_file_name
  [-supplemental supf_file_name]
  [-include_supply_exceptions]
  [-full_upf]
```

The `save_upf` command writes out a set of UPF commands to a specified file. The commands fully describe the power intent of the design in the same way that the commands written by the `write_sdc` command describe the timing constraints on a design.

Only synthesis and physical implementation tools make changes to the power design intent, so they are the only Synopsys tools that use the `save_upf` command. For example, the Design Compiler tool inserts isolation cells and level-shifter cells, and the IC Compiler or IC Compiler II tool inserts power switches. After making these types of changes, the tool can write out a new set of UPF commands that include the added power features. The generated UPF file can then be used by downstream tools.

The IC Compiler II and Fusion Compiler tools have options to control the content of the UPF file for specific purposes. For details, see the `save_upf` man page.

Synopsys analysis tools such as VCS, Formality, PrimeTime, and PrimePower do not change the power design intent, and therefore do not have the `save_upf` command.

Feedback

set_scope

```
set_scope [instance]
```

The `set_scope` command specifies the hierarchical scope of subsequent commands, including UPF commands. It has the same effect as the `current_instance` command. If no instance name is specified, the scope is set to the top level of the design. If the instance is specified as a single period character (.), the scope remains at the current instance. If the instance is specified as two period characters (..), the context is moved up one level in the instance hierarchy. If the instance string begins with a slash character (/), the scope changes to the instance whose name follows the slash character, relative to the top of the design.

The `set_scope` command reports the scope setting (a hierarchical path) before the setting is changed. This is different from the `current_instance` command, which reports the scope setting after the setting has been changed.

set_design_attributes

```
set_design_attributes
  [-models model_list]
  [-elements element_list]
  [-exclude_elements element_list]
  [-is_hard_macro {true|false}]
  {-attribute name value}*
```

*Asterisk indicates that the item in curly braces can be repeated in the command.

The `set_design_attributes` command sets the attributes of one or more cells.

The `-models` option specifies a list of design models on which the attributes are set. The `-elements` option specifies a collection of cells or supply sets on which the attributes are set. In the absence of the `-models` or `-elements` option, the attribute is set on the current top-level design.

The `-is_hard_macro` option specifies that the model is a hard macro. A hard macro might or might not have a UPF defined. If it is defined, then it must be self-contained.

The `-attribute` option specifies the *name* and *value* of the attributes to be set on the cells specified with the `-elements` option. The following attributes are supported:

```
... -attribute correlated_supply_group {true|false}
... -attribute derived_external {supply_set_name}
... -attribute derived_iso_strategy {isolation_strategy_name}
... -attribute external_supply_map {supply_set_name}
... -attribute lower_domain_boundary {true|false}
... -attribute merge_domain {true|false}
... -attribute reference_domain {true|false}
... -attribute suppress_iss {"create_power_domain ..."}
... -attribute upf_chip_design {true|false}
```

Feedback

```
... -attribute legacy_block {true|false}
... -attribute SNPS_default_power_upf2sv_vct {SV_TIED_HI SV_TIED_LO ...}
... -attribute contains_switches {list_of_switches}
... -attribute UPF_is_hard_macro {true|false}
```

The `-attribute` option supports both UPF version 2.0 and 2.1 syntax. You can specify the attribute in either of the following ways:

```
set_design_attributes -models my_model -attribute UPF_is_hard_macro true
set_design_attributes -models my_model \
                      -attribute (UPF_is_hard_macro true)
```

For more information about these attributes, see the `set_design_attributes` man page.

Feedback

set_port_attributes

```
set_port_attributes
  [-ports port_list]
  [-elements instance_names]
  [-exclude_elements]
  [-exclude_ports]
  [-applies_to inputs | outputs | both]
  [-attribute name value]*
  [-clamp_value 0 | 1 | latch]
  [-receiver_supply supply_set_reference]
  [-driver_supply supply_set_reference]
  [-repeater_supply supply_set_reference]
  [-model model]
  [-feedthrough]
  [-unconnected]
  [-is_analog]
  [-literal_supply supply_set_reference]
```

*Asterisk indicates that the item in square braces can be repeated in the command.

The `set_port_attributes` command specifies a collection of ports where the attributes must be set for the source or sink, for the interface of the power domains, when used with the `set_isolation` command; or to specify the connection properties of ports in a hierarchical model.

The `-ports` option specifies a collection of ports that are set as the source or sink for the `set_isolation` command, or the ports of a model.

The `-elements` option identifies a set of ports on the interface excluding the supply ports. If you specify a . (dot), the tool uses the current UPF scope. This is a required argument when the `-applies_to` option is specified.

The `-applies_to` option is used along with the `-elements` option. You can specify either `inputs`, `outputs`, or `both` with this option. When the `-applies_to` option is not specified, the tool assumes `both`.

The `-attribute` option specifies the *name* and *value* of the attributes to be set on the ports. The following attributes are supported:

```
... -attribute iso_sink {supply_set_name}
... -attribute iso_source {supply_set_name}
... -attribute snps_derived {true|false}
... -attribute related_supply_default_primary {true}
... -attribute repeater_power_net {supply_net_name}
... -attribute repeater_ground_net {supply_net_name}
```

For the IC Compiler II and Fusion Compiler tools, the additional attribute applies:

```
... -attribute resolved_iso_strategy {snps_none | strategy_name}
```

where the `strategy_name` is derived as *`DomainName_IsolationStrategyName`*. Use this attribute when you do not want the tool to insert an isolation cell on a port with an applicable source or sink strategy defined.

For more information about these attributes, see the `set_port_attributes` man page.

The `-attribute` option supports both UPF version 2.0 and 2.1 syntax. You can specify the attribute in either of the following ways:

```
set_port_attributes -elements B1/C1 \
    -attribute related_supply_default_primary true
set_port_attributes -elements B1/C1 \
    -attribute (related_supply_default_primary true)
```

The `-clamp_value` option specifies the clamp value for the specified ports. The value can be `0`, `1`, or `latch`. This option is equivalent to using the `-attribute` option to set the `UPF_clamp_value`.

The `-receiver_supply` option specifies the power supply for the receiving logic or the logic that fans out from the port. If the receiving logic is not within the logic design starting at the top level of the design, the tool assumes that the receiver supply is the supply for the receiving logic. You can specify the `-receiver_supply` option only at the top-level ports of the design.

The `-driver_supply` option specifies the power supply for the logic that drives the port. If the logic that drives the port is not within the logic that starts at the top level of the design, the tool assumes that the driver supply is the supply for the driver logic. You can specify the `-driver_supply` option only on the top-level ports of the design.

The `-repeater_supply` option specifies the supply set used by a repeater inserted to drive the port. A repeater is a buffer or inverter pair inserted in a long net to maintain the drive strength of the net.

The `-model`, `-feedthrough`, and `-unconnected` options specify the connection attributes of ports in a hierarchical model such as a library cell or macro cell. The `-model` option specifies the hierarchical model name and the `-ports` option specifies the ports of the model on which the attributes are set. The `-feedthrough` option specifies that there is an electrical short between the listed ports, or the `-unconnected` option specifies that the listed ports are unconnected and floating. The feedthrough or unconnected attribute applies to the listed ports in all instances of the specified model.

The `-exclude_elements` and `-exclude_ports` options exclude the listed objects or elements from the `set_port_attributes` command.

The `-is_analog` option specifies that the port is an analog port.

The `-literal_supply` option specifies the port's supply set. The specified port cannot be a hierarchical pin, macro input pin, or primary output pin.

Feedback

set_repeater

```
set_repeater strategy_name
  -domain domain_name
  [-elements element_list]
  [-exclude_elements element_list]
  [-applies_to input | output | both]
  [-applies_to_boundary upper | lower | both]
  [-repeater_supply supply_set_ref]
  [-name_prefix string]
  [-name_suffix string]
  [-update]
```

The `set_repeater` command specifies a strategy for inserting repeater cells (buffers) on the interface of a power domain. You can specify the supply for the repeater using this command. The `set_repeater` command replaces the `-repeater_supply` option of the `set_port_attributes` command.

Use the `-update` option to update the element list associated with the `-elements` and `-exclude_elements` options.

Use the -applies_to_boundary option to specify which boundary the strategy should apply. For more details, see Lower-Domain Boundary.

Query Commands

Each query commands returns a list of objects in the design that match the criteria you specify in the command.

find_objects

```
find_objects scope
  -pattern search_pattern
  -object_type object_type
  [-direction in | out | inout]
  [-transitive TRUE | FALSE]
  [-traverse_macros]
  [-regexp | -exact]
  [-non_leaf | -leaf_only]
  [-ignore_case]
```

The `find_objects` command finds instances, nets, or ports defined in the logic hierarchy in a specified scope and returns a list of object names that match a given search pattern.

The *scope* can be specified with a "." or a hierarchical scope. It cannot start with a ".."or "/" character. All elements returned are with respect to the current scope.

If the `-pattern` option is specified without the `-regexp` or `-exact` option, the search pattern is interpreted as a Tcl glob expression. With Tcl, using the "/" to match the

hierarchy separator is allowed, but it is not allowed if you use the `-regexp` option with the `-pattern` option. Similar to the scope, the search pattern cannot start with a ".."or "/" character.

The `-object_type` argument specifies the type of object to be found: `model`, `inst`, `port`, `supply_port`, or `net`. If you specify `model`, the command searches for instances of any model whose model name matches the search pattern. If you specify `inst`, `port`, or `net`, the command searches for objects of the specified type whose name matches the search pattern. If this option is not specified, instances, ports, nets, and processes are returned.

If you specify the object type to be `supply_port`, the command returns supply ports that match the search pattern. The tool filters the result based on other options such as the `-direction` and `-ignore_case` options. The `-transitive`, `-regexp`, and `-exact` options support the `supply_port` object type.

If you specify `port`, you can optionally restrict the search to ports having a specified direction by setting the `-direction` option to `in`, `out`, or `inout`.

By default, the search applies only to objects at the level of the specified scope. To extend the search to lower levels of hierarchy below that scope, use the `-transitive TRUE` option. A transitive search stops at a leaf cell boundary. To descend and traverse all hierarchies without stopping at any boundary marked as a soft macro or terminal boundary, use the `-traverse_macros` option. If you specify this option, the `-transitive` option is implicitly set to `true`.

By default, both hierarchical and leaf-level objects are returned by the command. To return only leaf-level objects or only hierarchical objects, use the `-leaf_only` or `-non_leaf` option, respectively.

Use the `-exact` option with the `-pattern` argument to specify a literal search string. Any wildcard characters specified in the string are not expanded when you use the `-exact` option.

The `-regexp` option indicates that the specified search pattern should be interpreted as a regular expression and not a Tcl glob expression.

The `save_upf` command does not write out the `find_objects` command in the output UPF, but instead resolves the objects.

query_cell_instances

```
query_cell_instances cell_name
  [-domain domain_name]
```

The `query_cell_instances` command returns a list of instance names for all instances of a given reference cell in the current scope of the design. You can optionally restrict the query to a given power domain.

Feedback

query_cell_mapped

```
query_cell_mapped instance_name
```

The `query_cell_mapped` command returns the reference cell name of a given cell instance.

query_map_power_switch

```
query_map_power_switch switch_name
  [-detailed]
```

The `query_map_power_switch` command returns the full `map_power_switch` command previously used to map library cells to a specified power switch. The `-detailed` option returns the same information in the form of a Tcl list of keyword and value pairs. If you specify an asterisk for the switch name, the command returns the mapping information for all power switches. It returns a warning message for each switch that is not yet mapped.

query_net_ports

```
query_net_ports net_name
  [-transitive TRUE | FALSE]
  [-leaf]
```

The `query_net_ports` command returns a list of the ports logically connected to a specified net. By default, the command returns only the ports present at the level of the current scope. To extend the search to lower levels of hierarchy below that scope, use the `-transitive TRUE` option. To return only leaf-level ports, use the `-leaf` option.

query_port_net

```
query_port_net port_name
  [-conn low | high]
```

The `query_port_net` command returns the name of the net logically connected to a specified port, if any such net exists.

A hierarchical port can be connected to nets having different names at the higher and lower levels of hierarchy. By default, the command returns the name of the higher-level connected net. To report the name of the lower-level net instead, use the `-conn low` option.

query_port_state

```
query_port_state port_name
  [-state state_name]
  [-detailed]
```

The `query_port_state` command returns information about the port states that have been previously defined for a specified port. The command "`query_port_state` *`port_name`*," without any other options, returns a list of the defined port states. If you use the `-state` *`state_name`* option, the command returns the full `add_port_state` command previously used to create that state for the port. The `-detailed` option returns the same information in the form of a Tcl list of keyword and value pairs. You can use the "`-state *`" option to return the commands used to create all of the port states.

query_power_switch

```
query_power_switch switch_name
  [-detailed]
```

The `query_power_switch` command returns information about previously defined power switches. The command "`query_power_switch *`" returns a list of the power switch names. If you specify the name of the particular power switch, the command returns the full `create_power_switch` command previously used to create that power switch. The `-detailed` option returns the same information in the form of a Tcl list of keyword and value pairs.

query_pst

```
query_pst table_name
  [-detailed]
```

The `query_pst` command returns information about previously defined power state tables in the active scope. The command "`query_pst *`" returns a list of the power state table names. If you specify the name of the particular power state table, the command returns the full `create_pst` command previously used to create that power state table. The `-detailed` option returns the same information in the form of a Tcl list of keyword and value pairs.

query_pst_state

```
query_pst_state state_name
  -pst table_name
  [-detailed]
```

The `query_pst_state` command returns information about the states that have been previously defined for a specified power state table. The command "`query_pst_state` *`-pst table_name`" returns a list of the power states defined for the specified table. If you specify the name of the particular power state, the command returns the full `add_pst_state` command previously used to create that power state. The `-detailed` option returns the same information in the form of a Tcl list of keyword and value pairs.

Feedback

Simulation/Verification Extension Commands

The simulation and verification extension commands support low-power checking by compatible tools.

bind_checker

```
bind_checker instance_name
  -module checker_name
  [-elements element_list]
  [-ports {{port_name net_name}}]
  [-parameters {{param_name param_value}}]
```

The `bind_checker` command inserts checker modules into the design for checking purposes, without affecting the design functionality, based on the mechanism of the SystemVerilog bind directive.

describe_state_transition

```
describe_state_transition transition_name
  -object object_name
  -from {from_list} -to {to_list} -paired {{from_state to_state}}
  [-legal | -illegal]
```

The `describe_state_transition` command specifies whether a transition of an object from one power state to another is legal. The `-from` and `-to` arguments specify many-to-many transitions. The `-paired` argument specifies one or more one-to-one transitions.

set_design_top

```
set_design_top module_or_hierarchical_instance
```

The `set_design_top` command specifies the top-level design instance. This information is used only by simulation and verification tools. The synthesis and physical implementation tools ignore this command and pass it on to downstream tools.

set_partial_on_translation

```
set_partial_on_translation
  [FULL_ON | OFF | UNDETERMINED]
  [-full_on_tools tool_list]
  [-off_tools tool_list]
```

The `set_partial_on_translation` command specifies the translation of the PARTIAL_ON state to either FULL_ON or OFF for purposes of evaluating the power state of supply sets and power domains. The synthesis and physical implementation tools ignore this command and pass it on to downstream tools.

Feedback

Supply Sets

A supply set is an abstract collection of supply nets, consisting of two supply functions, power and ground. A supply set is domain-independent, which means that the power and ground in the supply set are available to be used by any power domain defined within the scope where the supply set was created. However, each power domain can be restricted to limit its usage of supply sets within that power domain.

You can use supply sets to define power intent at the RTL level, so you can synthesize a design even before you know the names of the actual supply nets. A supply set is an abstraction of the supply nets and supply ports needed to power a design. Before such a design can physically implemented (placed and routed), its supply sets must be refined, or associated with actual supply nets.

Creating and Using Supply Sets

This is the UPF command syntax to create a supply set:

```
create_supply_set supply_set_name
 [-function {function_name supply_net_name}]*
 [-update]
```

*Asterisk indicates that the item in square braces can be repeated in the command.

You specify the name of the supply set and optionally specify the actual supply nets associated with the supply set. Typically, you initially create a supply set without specifying the supply nets. Later, you refine the supply set by executing the command again, but using the `-function` and `-update` options to associate the actual supply nets to the supply set.

For example,

```
prompt> create_supply_set SS1
SS1
prompt> create_supply_set SS2
SS2
```

These commands create supply sets called SS1 and SS2. Any power domain at the same or lower scope (hierarchical level) can use these power supplies by creating an association between the supply set name and the power domain's *supply set handles*. For example,

```
prompt> create_power_domain PD1 -elements {U1 U2 U3}
PD1
prompt> associate_supply_set SS1 -handle PD1.primary
1
```

This `associate_supply_set` command makes the SS1 supply set the primary supply of power domain PD1, so that the SS1 power and ground are the primary supplies of the domain. For more information about supply set handles, see Supply Set Handles.

If you already know which supply set is to be used for a new power domain, you can create the association when you create the domain. For example,

```
prompt> create_power_domain PD1 -elements {U1 U2 U3} \
         -supply {primary SS1}
```

You can use the notation “SS1.power” to mean the power supply function of the SS1 supply set, which is an abstract equivalent of a supply net. For example, the following command connects the “SS1.power” supply net to the VDD port:

```
prompt> connect_supply_net SS1.power -ports {VDD}
```

The Design Compiler tool can perform logic synthesis with the power supplies specified by supply sets. However, before you can perform physical implementation, you must refine the supply sets. Refinement means associating actual supply nets to the supply sets.

For example, the following command creates an association between the SS1 supply set functions and the VDD1a and VSS1a supply nets:

```
prompt> create_supply_set SS1 \
         -function {power VDD1a} \
        -function {ground VSS1a} \
        -update
```

This effectively makes the power function of SS1 the same as the VDD1a supply net, and similarly for the ground function of SS1 and the VSS1a supply net. If the supply set already exists, you must use the `-update` option with the `-function` options in the command.

After you make this association, you can refer to a supply net either by its net name or by its equivalent supply set function; “VDD1a” is the same as “SS1.power,” and “VSS1a” is the same as “SS1.ground”.

Per-Domain Supply Set Availability

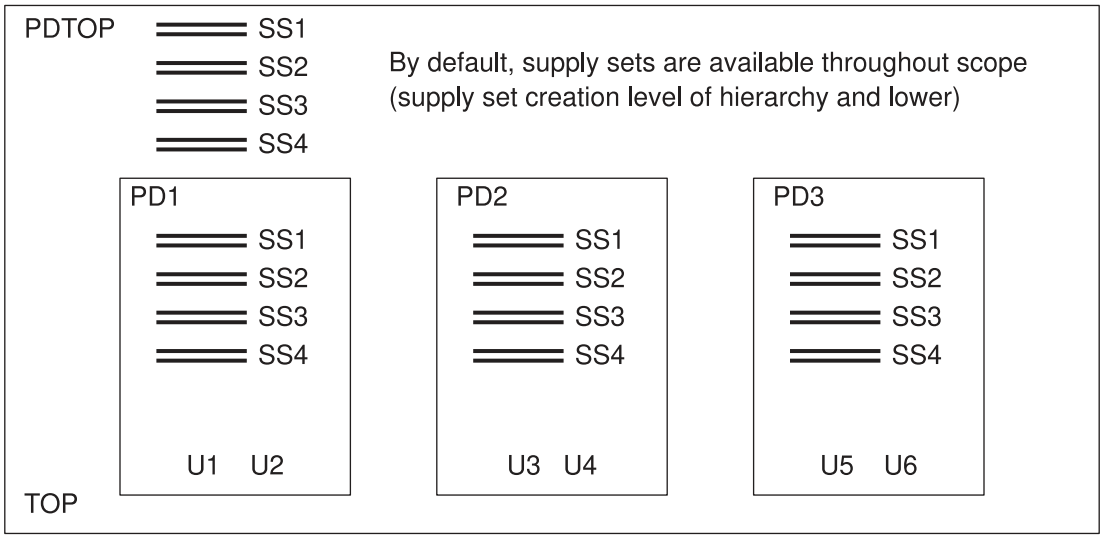
A supply set is domain-independent, which means that by default, the power and ground functions of the supply set are available to be used by any power domain defined within the scope where the supply set was created, for any purpose: primary supply, isolation, retention, level shifter insertion, and always-on synthesis. However, you can guide supply set usage by specifying which supply sets are allowed to be used within each power domain.

For example, consider the design shown in Figure 30. At the top level of the design, four supply sets are defined, SS1 through SS4. The power domain PDTOP includes all logic

Feedback

outside of blocks U1 through U6. Blocks U1 through U6 belong to three power domains: PD1, PD2, and PD3. The supply sets are domain-independent. By default, they are available to be used in the entire hierarchical level of TOP and lower levels.

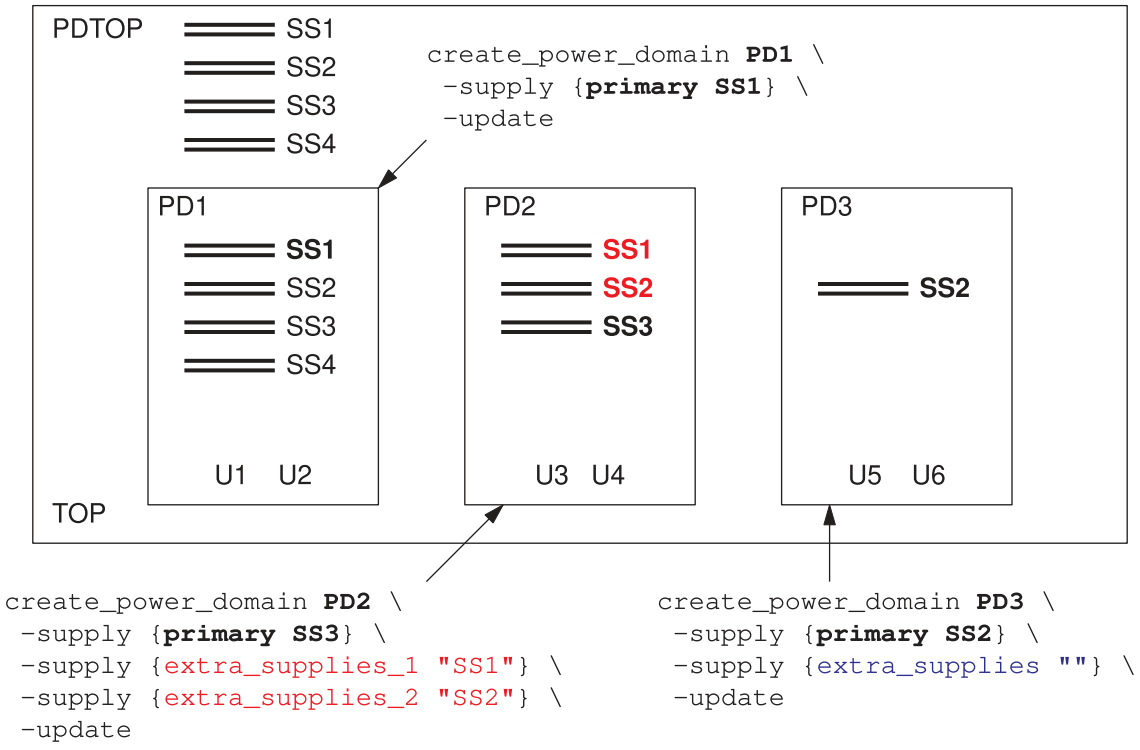
Figure 30 Default Supply Set Availability



However, suppose that you want to restrict the supply set usage as shown in Figure 31. You can do so by using the commands shown in the figure.

Feedback

Figure 31 Modified Supply Set Availability



In the first figure, Figure 30, the power domains and supply sets are defined as follows:

```
set_scope TOP
create_supply_set SS1
create_supply_set SS2
create_supply_set SS3
create_supply_set SS4
create_power_domain PDTOP -include_scope
create_power_domain PD1 -elements {U1 U2}
create_power_domain PD2 -elements {U3 U4}
create_power_domain PD3 -elements {U5 U6}
```

Feedback

In the second figure, Figure 31, the `-supply` option of the `create_power_domain` command guides the usage of supply sets in the power domains as follows:

- In PD1, only supply set SS1 can be used as the primary supply. The other supply sets are still available to be used for isolation, retention, level shifter insertion, and always-on synthesis.
- In PD2, only supply set SS3 can be used as the primary supply, and supply sets SS1 and SS2 are designated as *extra supplies*, which are the supply sets available to be used for lever shifter insertion and always-on synthesis.
- In PD3, only supply set SS2 can be used as the primary supply, and the keyword `extra_supplies`, used with the empty string, prevents the usage of any extra supplies.

To guide the usage of supply sets, use the following options of the `create_power_domain` command:

```
create_power_domain domain_name
 -supply {primary supply_set}            ; primary supply set
 -supply {default_isolation supply_set} ; default isolation supply set
 -supply {default_retention supply_set} ; default retention supply set
 -supply {extra_supplies_1 supply_set}  ; extra supply sets for level
 -supply {extra_supplies_2 supply_set}  ;  shifter insertion and
 -supply {extra_supplies_n supply_set}  ;  always-on synthesis
 -supply {extra_supplies ""}            ; no extra supply sets
 -update
```

A specified *default isolation supply set* is used when an isolation strategy is applied to the power domain, and an isolation power or ground supply is not specified in the strategy.

Similarly, a specified *default retention supply set* is used when a retention strategy is applied to the power domain, and a retention power or ground supply is not specified in the strategy.

When you define any extra supplies for a domain using the `extra_supplies_#` keyword (where each `#` is a unique numeral within the command), the domain is restricted to use only the following supplies:

- The primary supply of the domain
- The default retention supply for the domain
- The default isolation supply for the domain
- Any supplies specified in the retention strategies for the domain
- Any supplies specified in the isolation strategies for the domain
- Any domain-dependent supplies declared or reused in the domain
- Any extra supplies defined with the “`extra_supplies_#`” syntax

Feedback

The same restrictions apply when you use the `extra_supplies ""` syntax, except that no extra supplies can be used. You cannot use the `extra_supplies ""` syntax together with the `extra_supplies_n` keyword.

To create a domain-dependent supply net, use the `create_supply_net` command with the `-domain` option. This causes the supply net to be used only for a specific power domain. For example,

```
prompt> create_supply_net SN2 -domain PD2
```

Conversely, to create a domain-independent supply net, omit the `-domain` option. For example,

```
prompt> create_supply_net SN2
```

Refining Supply Sets to Supply Nets

A supply set is an abstraction of the supply nets and supply ports needed to power a design. Before such a design can be physically implemented (placed and routed), its supply sets must be refined, or associated with actual supply nets.

To refine an existing supply set, use the `create_supply_set` command with the `-function` option to specify the function and associated supply net, along with the `-update` option to update the supply set. For example,

```
prompt> create_supply_set SS1 \
         -function {power VDD1a} \
        -function {ground VSS1a} \
        -update
```

This effectively makes the power function of SS1 the same as the VDD1a supply net, and similarly makes the ground function of SS1 the same as the VSS1a supply net. Each supply net must be domain-independent and must exist at the same hierarchical scope as the supply set. Only one update is allowed for each function of a supply set.

Instead of specifying the supply nets directly, you can specify them indirectly through another supply set. For example,

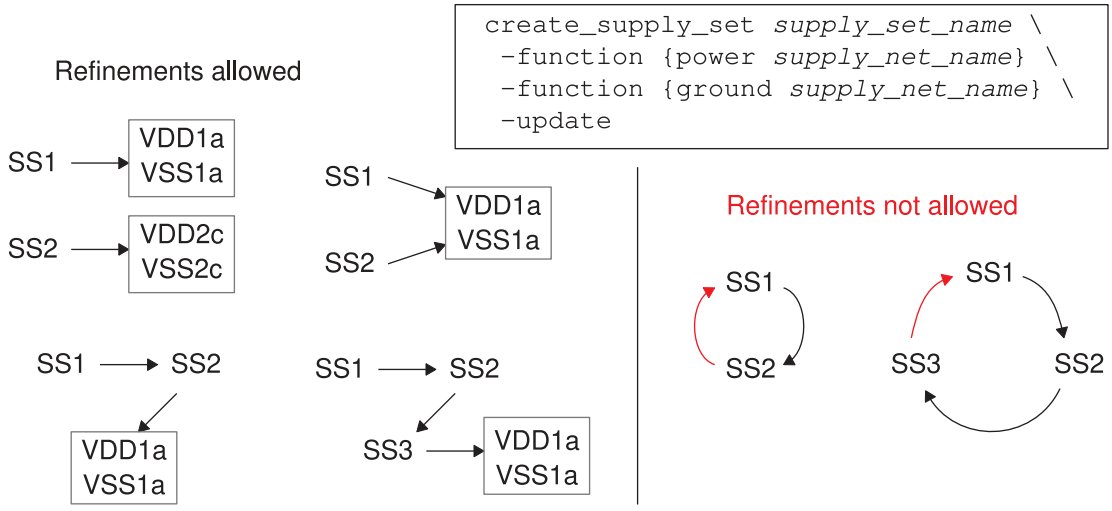
```
prompt> create_supply_set SS1 \
         -function {power SS2.power} \
        -function {ground SS2.ground} \
        -update
SS1
prompt> create_supply_set SS2 \
         -function {power VDD1a} \
        -function {ground VSS1a} \
        -update
SS2
```

Feedback

In this example, the functions of supply set SS1 are refined in terms of supply set SS2. Supply set SS2 is then refined directly to actual supply nets, so the two supply sets are refined to the same supply nets. After this refinement, you can refer to the power supply function of SS1 equivalently as SS1.power, SS2.power, or VDD1a.

Multiple supply sets can be refined to the same supply nets, either directly or indirectly. However, all refinements must be eventually resolved to actual supply nets, so circular refinements are not allowed. See Figure 32.

Figure 32 Refinement of Multiple Supply Sets to the Same Supply Nets



To model a shared or common ground between two supply sets, create a domain-independent supply net and update both supply set handles to use the same supply net for the ground function. For example,

```
create_power_domain TopPD
create_power_domain PD1 -elements {Block1}
create_supply_net VSS1
create_supply_set TopPD -function {ground VSS1} -update
create_supply_set PD1 -function {ground VSS1} -update
```

Supply Sets in Hierarchical Flows

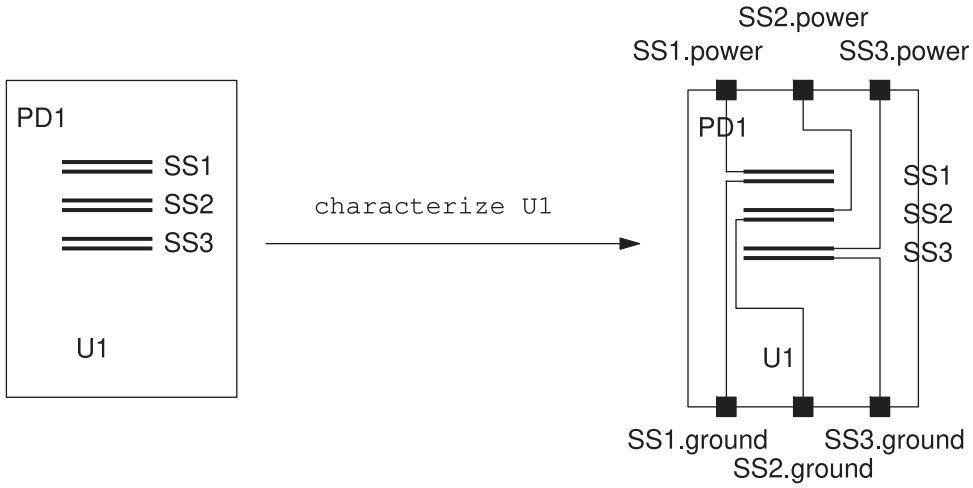
Using supply sets lets you perform logic synthesis even before you create supply ports or know the names of the actual supply sets. The supply sets do not need to be refined until you reach the physical implementation stage in the IC Compiler, IC Compiler II, or Fusion Compiler tool.

In a hierarchical flow, when a block is characterized or partitioned, the Design Compiler tool automatically creates supply ports for the power and ground functions of each supply

Feedback

set to maintain power to the block from the higher level of hierarchy. For example, for a block with three supply sets, the Design Compiler tool creates six supply ports, as shown in Figure 33.

Figure 33 Supply Ports Created During Characterization



Each new supply port created by this process has an attribute called `snps_derived` to identify it as a tool-derived supply port. This attribute allows implementation tools to tell the difference between tool-generated and user-defined ports. The tool sets the attribute by running this command in the background:

```
set_port_attributes -ports new_port_name -attribute snps_derived true
```

When a tool-created block is merged with the top-level block, the `propagate_constraints` command removes the tool-generated supply ports, as identified by the `snps_derived` attribute setting, to optimize the design. Supply nets connected to each other through the supply port are considered to be the same supply net.

If the supply sets are already refined to supply nets before characterization, then the supply ports created for the supply nets are not automatically removed. Instead, they are retained and used to connect the supply nets in the block to the top level. If multiple supply sets are refined to the same supply net, the tool creates just one supply port for each unique supply net, possibly spanning multiple supply sets.

Supply Set Handles

A *supply set handle* is an abstract supply set implicitly created for a power domain. A newly created power domain has supply set handles for the domain's primary supply set,

Feedback

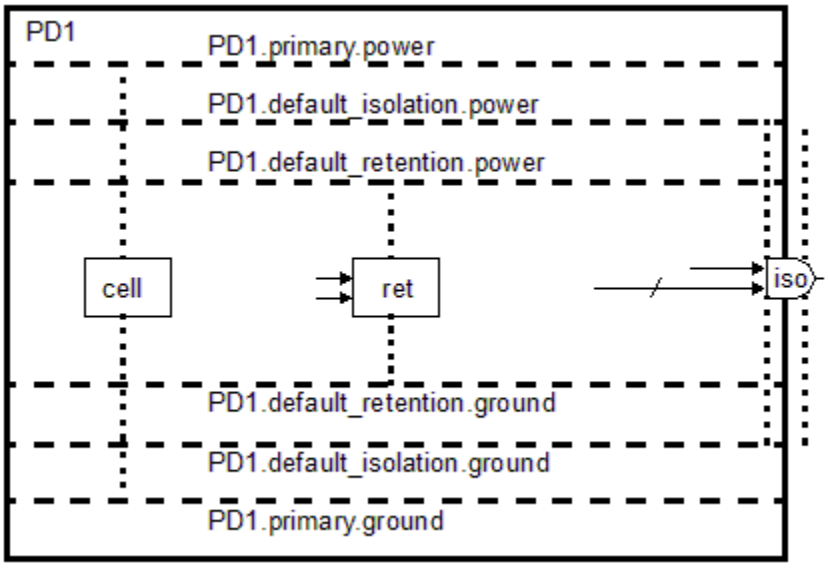
a default isolation supply set, and a default retention supply set. These supply set handles let you synthesize a design even before you create any supply sets, supply nets, and supply ports for the power domain. Before such a design can physically implemented, its supply set handles must be *refined*, or associated with actual supply sets; and those supply sets must be refined so that they are associated with actual supply nets.

Creating a new power domain with the `create_power_domain` command implicitly creates the following supply set handles:

- `primary` – The primary supply set of the power domain.
- `default_isolation` – The default isolation supply set for any isolation strategy applied to this power domain, which is used when the isolation power or ground is not otherwise specified in the strategy.
- `default_retention` – The default retention supply set for any retention strategy applied to this power domain, which is used when the retention power or ground is not otherwise specified in the strategy.

For example, when you use the `create_power_domain` command to create a power domain called `PD1`, the command implicitly creates the supply set handles `PD1.primary`, `PD1.default_isolation`, and `PD1.default_retention`, providing the implicit power supply nets shown in Figure 34.

Figure 34 Implicit Supply Set Handles



Feedback

Primary Supply Set Handle

The `create_power_domain` command automatically creates a primary supply set handle for the domain. All cells in the power domain are considered connected to that supply set handle, so it serves as the default primary supply for the domain.

For example, consider the following two `create_power_domain` commands:

```
create_power_domain Top_PD -include_scope
create_power_domain BLOCK_PD -elements {myBlock}
```

The first command creates the `Top_PD` power domain and a supply set handle called `primary`. All cells in the power domain are considered connected to the `Top_PD.primary` supply set. The primary supply net is `Top_PD.primary.power` and the primary ground net is `Top_PD.primary.ground`.

The second command creates a power domain called `BLOCK_PD`, which includes the block `myBlock`, and creates a supply set handle called `primary` for the new domain. The `myBlock` block no longer belongs to the `Top_PD` power domain. All the lower-level cells in `myBlock` are considered connected to the `BLOCK_PD.primary` supply set. Within the `BLOCK_PD` power domain, the primary supply net is `BLOCK_PD.primary.power` and the primary ground net is `BLOCK_PD.primary.ground`.

Isolation Supply Set Handle

The `create_power_domain` command automatically creates a default isolation supply set handle for the domain. This supply set handle is used for any isolation strategy that does not explicitly specify an isolation power supply.

For example, consider the following commands:

```
create_power_domain TOP
create_power_domain BlockPD -elements Block1

set_isolation blk_iso_out -domain BlockPD \
 -clamp_value 1 -applies_to outputs
set_isolation_control blk_iso_out -domain BlockPD \
 -isolation_signal iso_ctrl -isolation_sense high -location parent
```

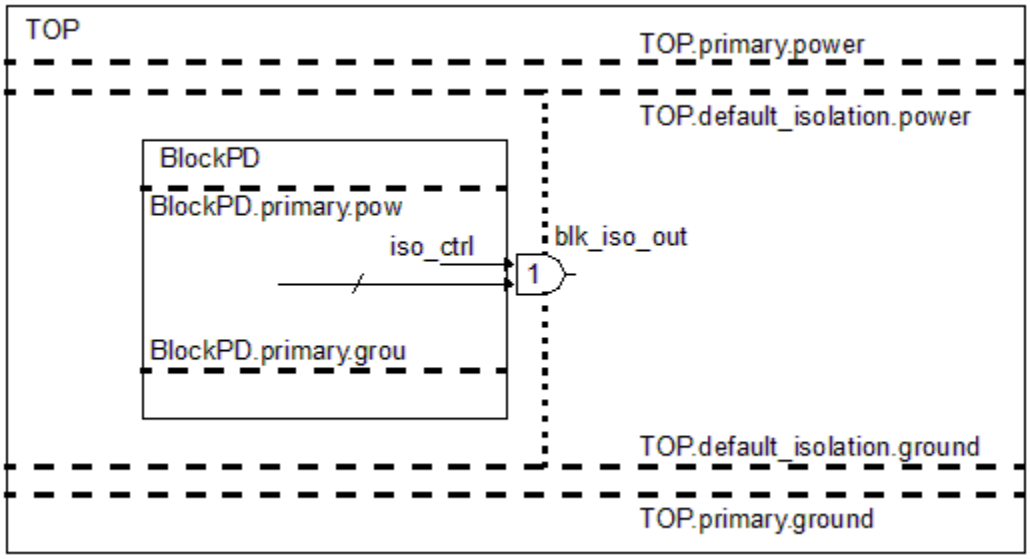
The first `create_power_domain` command creates the `TOP` power domain, which includes all elements in the current scope, and also creates a supply set handle called `default_isolation`. The second `create_power_domain` command creates the `BlockPD` power domain, which includes the `Block1` block, and also creates a supply set handle called `default_isolation` for that domain.

The `set_isolation` command creates an output isolation strategy for the `BlockPD` power domain. The `set_isolation_control` command defines the isolation control signal and the location of the isolation cells for the strategy. The `set_isolation` command does

Feedback

not specify the isolation supply set or supply nets, and the `set_isolation_control` command sets the location to `parent`, so the output isolation cells use the default isolation supply set handle of the parent domain to drive the fanout of the isolation cell, as shown in Figure 35.

Figure 35 Default Isolation Power Supplies



The strategy implicitly connects the isolation cell to the supplies of the default isolation supply set handle. These supplies are `TOP.default_isolation.power` and `TOP.default_isolation.ground`. It is not necessary to use the `-isolation_power_net`, `-isolation_ground_net`, or `-isolation_supply` options of the `set_isolation` command.

Alternatively, in the `set_isolation` command, you can explicitly specify the supply sets to use for the isolation source and sink. For example,

```
create_power_domain TOP
create_power_domain BlockPD -elements Block1

set_isolation blk_iso_out -domain BlockPD \
 -source BlockPD.primary -sink TOP.primary
 -clamp_value 1 -applies_to outputs
set_isolation_control blk_iso_out -domain BlockPD \
 -isolation_signal iso_ctrl -isolation_sense high -location self
```

Note:

Synopsys tools allow the `-applies_to` option to be used together with the `-source` and `-sink` options in the same `set_isolation` command. However, this usage is not supported in the IEEE 1801 standard. Therefore, if you plan to

use the UPF file with third-party tools, you should use an alternative method of specifying the source, sink, and applies-to aspects of the isolation strategy.

In this example, because the isolation location is specified as `self`, the isolation strategy calls for the isolation cell to be located in the `BlockPD` instead of the parent level, `TOP`. Otherwise, the isolation strategy is the same as in the previous example, shown in Figure 35.

Retention Supply Set Handle

The `create_power_domain` command automatically creates a default retention supply set handle for the domain. This supply set handle is used for any retention strategy that does not explicitly specify a retention power supply.

For example, consider the following commands:

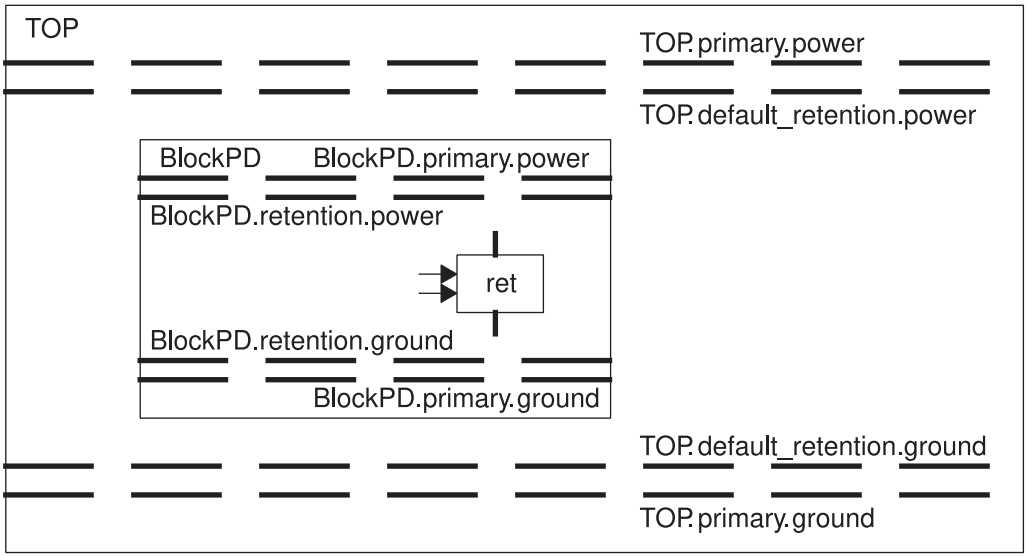
```
create_power_domain TOP
create_power_domain BlockPD -elements Block1
set_retention block_ret -domain BlockPD
set_retention_control block_ret -domain BlockPD \
 -save_signal {Block1/all_save high} \
 -restore_signal {Block1/all_restore low}
```

The first `create_power_domain` command creates the `TOP` power domain, which includes all elements in the current scope. The second `create_power_domain` command creates the `BlockPD` power domain, which includes the `Block1` block, and also creates a supply set handle called `default_retention` for that domain.

The `set_retention` command creates a retention strategy for the `BlockPD` power domain. The `set_retention_control` command specifies the save and restore signals for the strategy. The `set_retention` command does not specify the retention supply set or supply nets, so the output retention cells use the default retention supply set handle, as shown in Figure 36.

Feedback

Figure 36 Default Retention Power Supplies



The strategy implicitly connects retention cells to the supplies of the default retention supply set handle. These supplies are `BlockPD.default_retention.power` and `BlockPD.default_retention.ground`. It is not necessary to use the `-retention_power_net`, `-retention_ground_net`, or `-retention_supply` options of the `set_retention` command.

User-Specified Supply Set Handles

In addition to the implicit supply set handles created for primary power, isolation, and retention, you can also create your own supply set handles by using the `-supply` option of the `create_power_domain` command. For example,

```
prompt> create_power_domain PD1 -elements {U1 U2} -supply {my_handle}
```

This creates a supply set handle named `my_handle`. In this example, you can reference the supply set as `PD1.my_handle` and the supply nets as `PD1.my_handle.power` and `PD1.my_handle.ground`. For example,

```
set_retention block_ret -domain BlockPD \
 -retention_supply PD1.my_handle
```

or

```
set_retention block_ret -domain BlockPD \
 -retention_power_net PD1.my_handle.power \
 -retention_ground_net PD1.my_handle.ground
```

Feedback

The following supply set handle names are reserved for special purposes:

- `primary` – The primary supply set of the power domain.
- `default_isolation` – The default isolation supply set for the power domain.
- `default_retention` – The default retention supply set for the power domain.
- `extra_supplies_#` (where `#` represents an integer) – Extra supplies that are available to be used for level shifter insertion and always-on synthesis.
- `extra_supplies ""` – Causes the power domain not to use any extra supplies.

Preventing Automatic Creation of Supply Set Handles

If you do not want supply set handles to be created automatically for each new power domain, execute the following command before you create any power domains:

```
prompt> set_app_var upf_create_implicit_supply_sets false
```

After you set this variable to `false`, the `create_power_domain` command does not create the `primary`, `default_isolation`, and `default_retention` supply set handles unless you explicitly use the `-supply` option.

Note:

This option is not available in the IC Compiler II or Fusion Compiler tool.

After you create the first power domain, the variable becomes read-only and cannot be changed. Therefore, you need to set the variable before you create any power domains.

If you want to prevent automatic creation of supply set handles for some power domains but not others, leave the `upf_create_implicit_supply_sets` variable set to `true`. Instead, use the `set_design_attributes` command's `suppress_iss` attribute to list the power domains that you do not want to use supply set handles. For example,

```
prompt> set_design_attributes -attribute suppress_iss "myBLOCK_PD"
```

This example prevents automatic supply set handle creation for the power domain called `myBLOCK_PD`. This is useful when you are using newer blocks that use supply set handles along with older blocks that do not. To connect the power supplies to the supply ports of such older blocks, use the `connect_supply_net` command. For example,

```
set_design_attributes -attribute suppress_iss "myBLOCK_PD"
# Power domain definition
create_power_domain Top_PD -supply {aolow}
create_power_domain myBLOCK_PD -elements {myBlock}
# Load block UPF, no supply set handles for myBLOCK_PD domain
load_upf -scope myBlock myBlock.upf
# Connect supply set to block ports
```

Feedback

```
connect_supply_net Top_PD.primary.power -ports Block/AOVDD
connect_supply_net Top_PD.primary.ground -ports Block/VSS
```

Supply Set Handle Refinement

Before a design can be physically implemented (placed and routed), the supply set handles must be refined, so that each handle is effectively connected to a supply set. Refinement is the process of associating a supply set function to a handle. In turn, the supply set function is itself must be refined to actual power and ground supply nets.

You can use the following commands to refine supply set handles:

```
create_supply_set
associate_supply_set
create_power_domain
set_domain_supply_net
```

When you use the `create_supply_set` command for this purpose, it creates a supply set for the handle and directly associates that supply set with actual power and ground supply nets. On the other hand, the `associate_supply_set` command creates an association between a supply set handle and a supply set (or possibly another supply set handle), without refining the supply to actual power and ground supply nets. The final refinement of the supply set to supply nets can be done later, as described in Refining Supply Sets to Supply Nets. You can also make the association immediately when you create a power domain with the `create_power_domain` command. For the primary supply set handle only, you can also use the `set_domain_supply_net` command.

Handle Refinement Using create_supply_set

To refine a supply set handle directly to supply nets, use the `create_supply_set` command. For example,

```
create_supply_set PD1.primary \
 -function {power VDD1} \
 -function {ground VSS1) \
 -update
```

This creates a supply set for the `PD1.primary` supply set handle and also refines the supply set to specific supply nets. The `-update` option must be used in the command because the `PD1.primary` supply set handle already exists.

The argument of the `-function` option must be an actual supply net, not the function of another supply set. For example, the following indirect refinement is not allowed:

```
create_supply_set SS1
create_supply_set PD1.primary \
 -function {power SS1.power} \    ;not allowed
 -function {ground SS1.ground) \  ;not allowed
 -update
```

However, indirect refinement is possible by using the `associate_supply_set` command.

Handle Refinement Using associate_supply_set

To create an association between a supply set handle and a supply set, use the `associate_supply_set` command. For example,

```
create_supply_set SS1
create_power_domain PD1
associate_supply_set SS1 -handle PD1.primary
```

The `associate_supply_set` command effectively creates a connection between the `SS1` supply set and the `PD1.primary` supply set handle. Before physical implementation can be performed, the `SS1` supply set must be refined to actual supply nets.

Each supply set handle can be associated only one time, so be sure of your association before you execute the `associate_supply_set` command.

When making an association, the supply set handle must be in the scope of the supply set. In other words, the supply set handle must exist at the level of hierarchy where the supply set was created or lower. A supply set handle can be associated with a higher-level (but not lower-level) supply set.

You can create an association indirectly through one or more other supply set handles. For example,

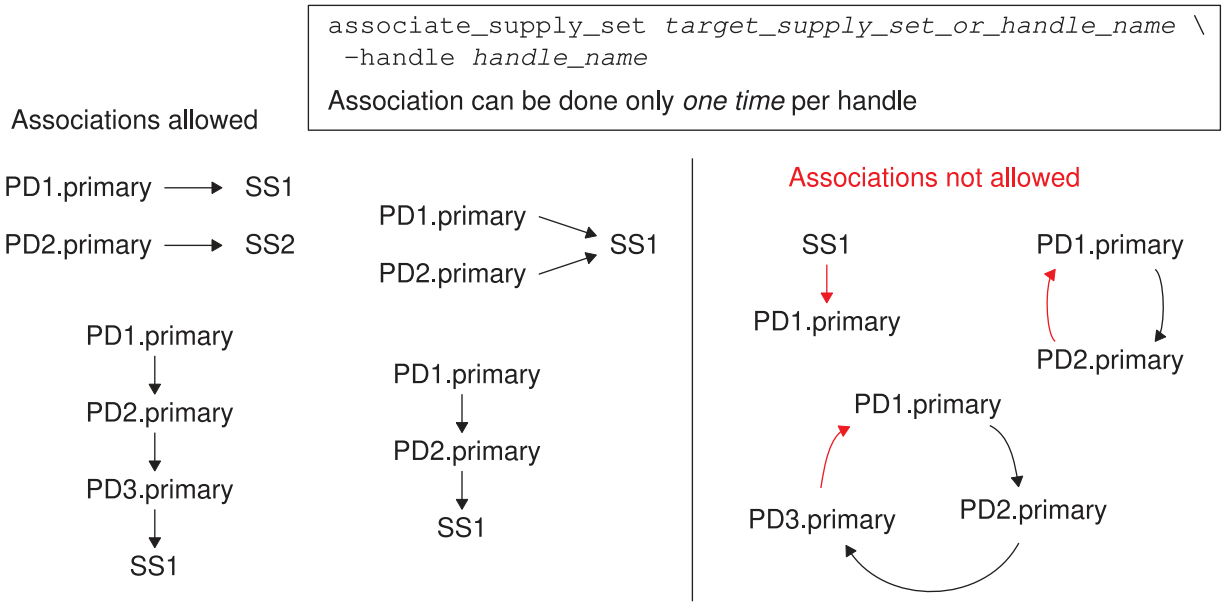
```
create_supply_set SS2
associate_supply_set SS2 -handle PD1.default_isolation
associate_supply_set PD1.default_isolation \
 -handle PD1.default_retention
```

In this example, the PD1 default retention strategy handle is refined to the PD1 default isolation strategy, which in turn is refined to the SS2 supply set. In this case, the isolation and retention strategies are both refined to supply set SS2 and share the same power characteristics.

Multiple supply set handles can be refined to the same supply nets, either directly or indirectly. However, all refinements must be eventually resolved to actual supply sets, so circular refinements are not allowed. Also, associations are not allowed in the reverse direction, from a supply set to a supply set handle. See Figure 37.

Feedback

Figure 37 Association of Multiple Supply Set Handles to the Same Supply Set



Handle Refinement Using create_power_domain

You can associate a supply set handle to a supply set at the same time that you create the power domain by using the `-supply` option of the `create_power_domain` command. For example,

```
create_supply_set SS1
create_power_domain PD1 -elements {U1 U2} -supply {primary SS1}
```

Alternatively, you can create the power domain first, and then later refine the supply set handle to a specific supply set by executing the `create_power_domain` command again with the `-update` option. For example,

```
create_power_domain PD1 -elements {U1 U2}

...

create_supply_set SS1
create_power_domain PD1 -supply {primary SS1} -update
```

The `-update` option can be used only with the `-supply` option, and only for the purpose of associating the supply set handle to a supply set. It is not necessary (or allowed) to use the `-elements` option in the update. Also, you cannot change a supply set handle association, so be sure that the association is correct before you perform the update.

Feedback

Primary Handle Refinement Using set_domain_supply_net

For the *primary* supply set handle only (*domain_name*.primary), you can perform refinement directly to supply nets with the set_domain_supply_net command. For example,

```
create_supply_net myVDD
create_supply_net mvVSS
create_power_domain PD1 -elements {U1}
set_domain_supply_net PD1 \
 -primary_power_net myVDD \
 -primary_ground_net myVSS
```

The set_domain_supply_net command creates an implicit association between the primary supply set handle and the specified supply nets. However, verification tools cannot reliably recognize this type of association, so for better verification results, using one of the other refinement methods is recommended .

Supply Set Handle Usage Example

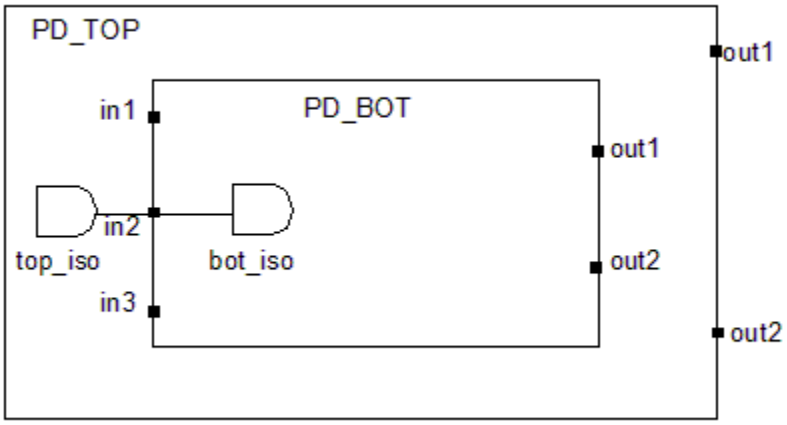
The following example demonstrates the usage of supply set handles.

```
## Power domain definition
create_power_domain TopPD -supply {aolow}
create_power_domain PD1 -elements {Block1}

# Association
associate_supply_set PD1.default_isolation \
-handle PD1.default_retention
associate_supply_set TopPD.aolow \
-handle PD1.default_isolation

# Power state definition
add_power_state TopPD.aolow \
 -state TOP_LV {-supply_expr {power == `{FULL_ON, 0.7}}}
add_power_state Top_PD.aolow \
 -state GND {-supply_expr {ground == `{FULL_ON, 0.0}}}
...

# Isolation and retention strategies
set_isolation iso1 -domain PD1 ...
set_isolation_control iso1 -domain PD1 ...
set_retention ret1 -domain PD1 ...
set_retention_control ret1 -domain PD1 ...
```

This script creates the power domains, power supply set handles, and isolation and retention strategies shown in Figure 38.

Feedback

Figure 38 Supply Set Handle Usage Example



The first `create_power_domain` command creates the top-level power domain, `TopPD`, and also creates a user-defined supply set handle called `aolow` (always-on low).

The two `associate_supply_set` commands associate the default retention and isolation supply set handles of power domain `PD1` with the `aolow` supply set handle of the top-level domain. The first command associates the `PD1.default_retention` handle to the `PD1.default_isolation` handle, while the second command associates the `PD1.default_isolation` handle to the `TopPD.aolow` handle.

Before physical implementation can be done, the `TopPD.aolow` supply set handle must be refined to actual supply nets.

Because of the associations between the supply set handles `PD1.default_retention`, `PD1.default_isolation`, and `TopPD.aolow`, all three of these handles represent the same power and ground supplies and share the same power state definitions. You can use any of the three supply set handle names to reference the same power supply set.

The `TopPD` supply sets are available to be used in domain `PD1` because both the `TopPD` and `PD1` domains were created in the same scope. Both were created at the top level, even though the *extent* of `PD1` is a lower-level block.

The following example demonstrates an association that is not allowed.

```
## Power domain definition
create_power_domain TopPD -supply {aolow}
set_scope Block1
create_power_domain PD1
set_scope ..

# Association
associate_supply_set Block1/PD1.default_isolation \
```

Feedback

```
 -handle PD1.default_retention          ;error, not allowed
associate_supply_set TopPD.aolow \
 -handle Block1/PD1.default_isolation  ;allowed
```

The target supply set (or target supply set handle) must be at or above the scope of the supply set handle specified by the `-handle` argument.

The same situation applies when you specify a lower-level scope with the `-scope` option of the `create_power_domain` command:

```
## Power domain definition
create_power_domain TopPD -supply {aolow}
create_power_domain PD1 -elements Block1 -scope Block1

# Association
associate_supply_set Block1/PD1.default_isolation \
 -handle PD1.default_retention          ;error, not allowed
associate_supply_set TopPD.aolow \
 -handle Block1/PD1.default_isolation  ;allowed
```

Supply Set Handles in Design Compiler Hierarchical Flows

The following example demonstrates the usage of supply set handles in a hierarchical flow, using the `characterize` command to characterize the power supply environment of a block instance and the `propagate_constraints` command to propagate power supply constraints from the lower level to the top level.

Suppose that you specify the power intent of a design using the following script:

```
create_power_domain TopPD -include_scope
create_power_domain PD1 -elements I_REG_FILE
associate_supply_set TopPD.primary -handle TopPD.default_isolation
set_isolation block_iso_out -domain PD1 -clamp_value 1 \
 -sink TopPD.primary -isolation_supply TopPD.default_isolation
set_isolation_control block_iso_out -domain PD1 \
 -isolation_signal enable_signal2 -isolation_sense high -location self
set_level_shifter myLS -domain PD1 -location self
```

This creates the top-level power domain, `TopPD`, and another power domain, `PD1`, for a lower-level block instance, `I_REG_FILE`. It also sets the isolation and level shifter strategies for `PD1`.

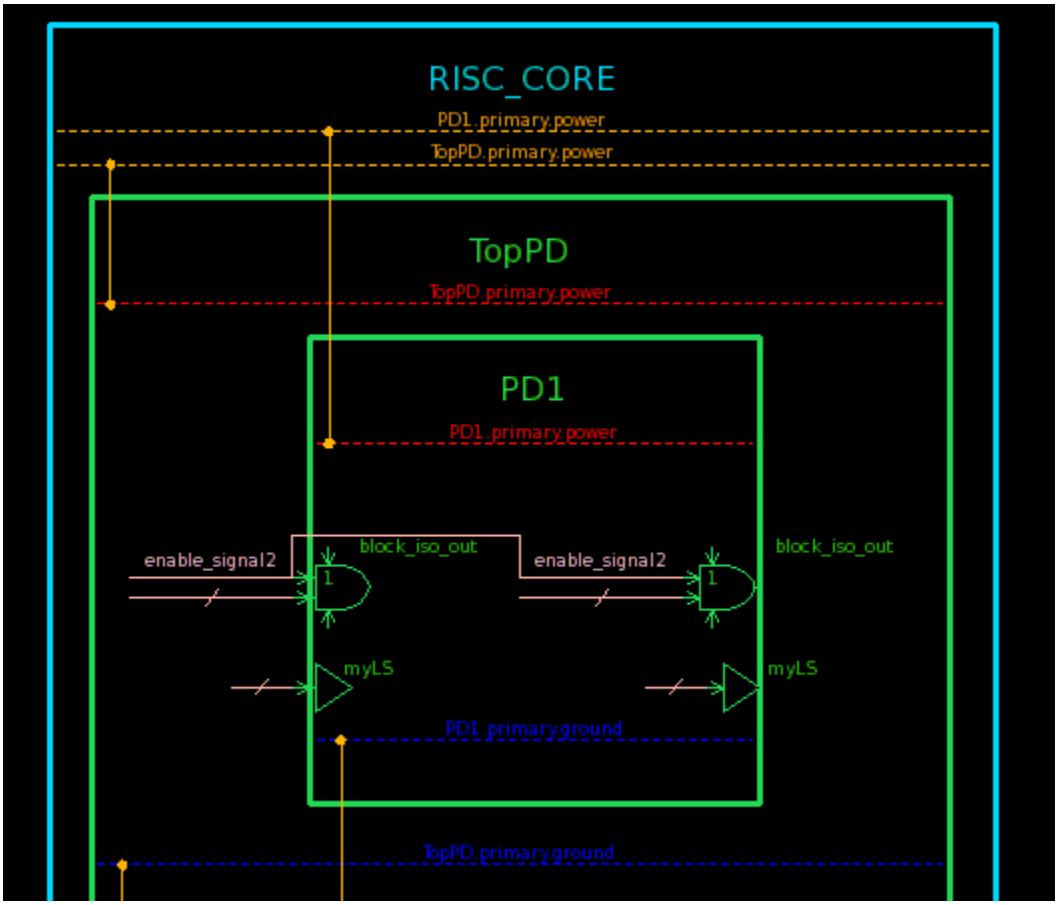
The `TopPD.default_isolation` supply set is associated with the `TopPD.primary` supply set, so that these two supply sets are combined into a single group. At the top-level scope, there are two groups: `TopPD.primary` (which also includes `TopPD.isolation`) and `PD1.primary`, the primary supply for the `I_REG_FILE` block.

In Design Vision, the GUI tool of Design Compiler, you can view the current UPF power intent in schematic diagram form, as shown in Figure 39. To display the diagram, start

Feedback

the GUI (`gui_start`) and choose Power > UPF Diagram > New UPF Diagram View. Alternatively, you can choose Power > Visual UPF, and click the Diagram tab.

Figure 39 Initial Power Intent Diagram



At this point, you can use the characterize command to characterize the power supply environment of the I_REG_FILE block:

```
prompt> list_instances
...
I_REG_FILE (REG_FILE)
...
prompt> characterize I_REG_FILE
  Characterizing cell 'I_REG_FILE' on design 'REG_FILE'
I_REG_FILE/PD1.primary
I_REG_FILE/PD1.default_isolation
I_REG_FILE/PD1.default_retention
1
prompt> save_upf post-char.upf
```

When you characterize the block, the Design Compiler tool automatically creates supply nets and supply ports for the power and ground functions of each supply set handle. The UPF file written out by the `save_upf` command shows the commands inserted by the tool (highlighted in blue in the following example):

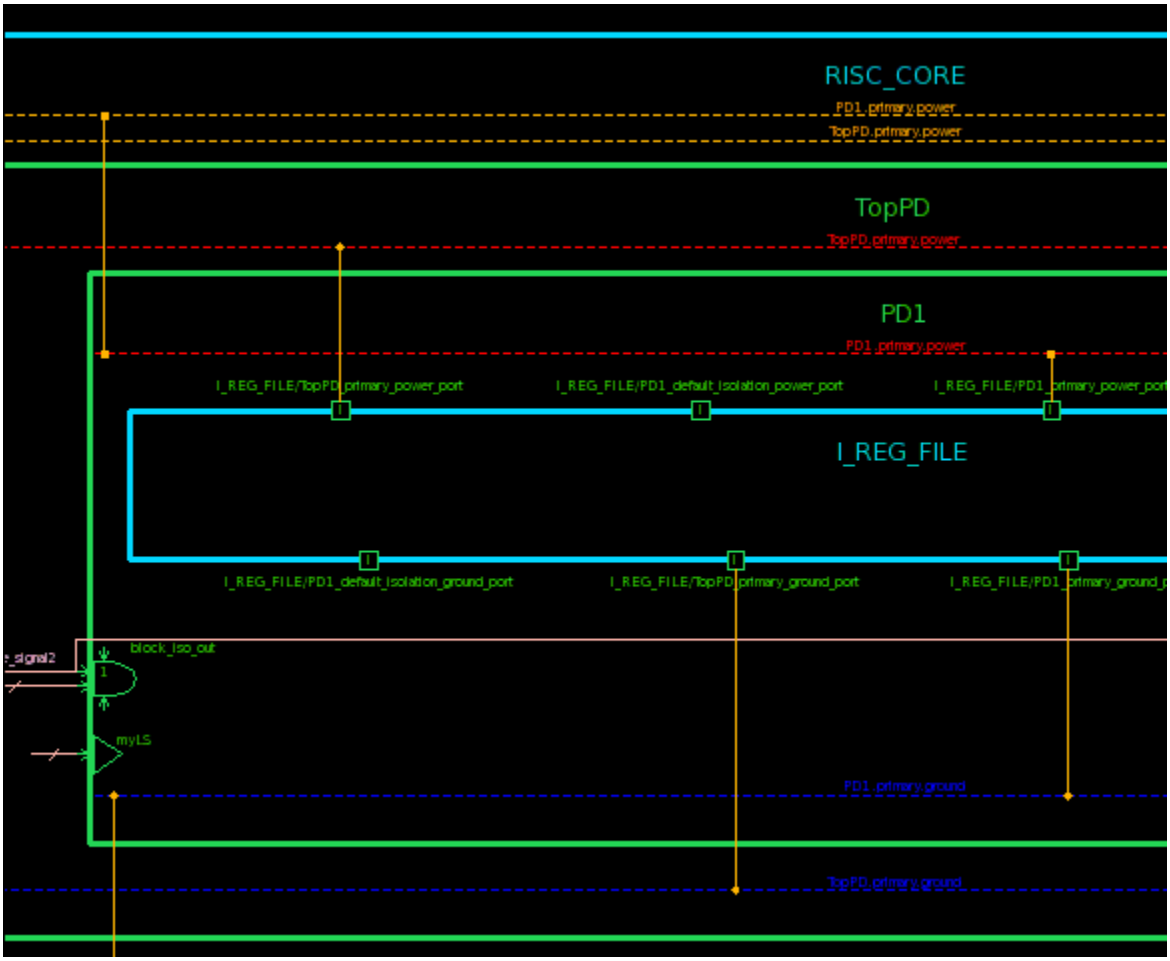
```
#Generated by Design Compiler ...
create_power_domain TopPD -include_scope
create_power_domain PD1 -elements I_REG_FILE
associate_supply_set TopPD.primary -handle TopPD.default_isolation
create_supply_port I_REG_FILE/TopPD_primary_power_port -direction in
create_supply_port I_REG_FILE/TopPD_primary_ground_port -direction in
create_supply_port I_REG_FILE/PD1_primary_power_port -direction in
create_supply_port I_REG_FILE/PD1_primary_ground_port -direction in
create_supply_port I_REG_FILE/PD1_default_isolation_power_port -direction
 in
create_supply_port I_REG_FILE/PD1_default_isolation_ground_port
 -direction in
create_supply_port I_REG_FILE/PD1_default_retention_power_port -direction
 in
create_supply_port I_REG_FILE/PD1_default_retention_ground_port
 -direction in
connect_supply_net TopPD.primary.power -ports
 I_REG_FILE/TopPD_primary_power_port
connect_supply_net TopPD.primary.ground -ports
 I_REG_FILE/TopPD_primary_ground_port
connect_supply_net PD1.primary.power -ports
 I_REG_FILE/PD1_primary_power_port
connect_supply_net PD1.primary.ground -ports
 I_REG_FILE/PD1_primary_ground_port
connect_supply_net PD1.default_isolation.power \
 -ports I_REG_FILE/PD1_default_isolation_power_port
connect_supply_net PD1.default_isolation.ground \
 -ports I_REG_FILE/PD1_default_isolation_ground_port
connect_supply_net PD1.default_retention.power \
 -ports I_REG_FILE/PD1_default_retention_power_port
connect_supply_net PD1.default_retention.ground \
 -ports I_REG_FILE/PD1_default_retention_ground_port
set_isolation block_iso_out -domain PD1 -isolation_supply \
 TopPD.default_isolation -clamp_value 1 -sink TopPD.primary
set_isolation_control block_iso_out -domain PD1 -isolation_signal \
 enable_signal2 -isolation_sense high -location self
set_port_attributes -ports {I_REG_FILE/TopPD_primary_power_port \
 I_REG_FILE/TopPD_primary_ground_port I_REG_FILE/PD1_primary_power_port \
 I_REG_FILE/PD1_primary_ground_port \
 I_REG_FILE/PD1_default_isolation_power_port \
 I_REG_FILE/PD1_default_isolation_ground_port \
 I_REG_FILE/PD1_default_retention_power_port \
 I_REG_FILE/PD1_default_retention_ground_port} \
 -attribute snps_derived TRUE
set_level_shifter myLS -domain PD1 -location self
```

Feedback

A `set_port_attributes` command sets the `snps_derived` port attributes to `TRUE` for the newly created ports. This attribute is an indicator that the ports were created by the tool rather than the user. The tool can later remove these ports during optimization.

In the Design Vision GUI, you can see the characterized block and the added supply nets and supply ports, as shown in Figure 40.

Figure 40 Power Intent Diagram After Characterization



After block characterization is complete, you can use the `propagate_constraints` command to propagate power supply constraints from the lower level to the top level:

```
prompt> propagate_constraints -design REG_FILE
#   Propagate Constraints from cell I_REG_FILE/ (REG_FILE) #
Info: Merging domains PD1 and I_REG_FILE/PD1
Removing Supply Set I_REG_FILE/TopPD_primary
Removing Supply Set I_REG_FILE/PD1_primary
Removing Supply Set I_REG_FILE/PD1_default_isolation
```

```
Removing Supply Set I_REG_FILE/PD1_default_retention
1
```

Constraint propagation reconciles top-level and block-level supply sets and merges any duplicate supply sets. It restores the supply set handle names used in strategy commands, using the first member of each supply set handle group. The name chosen from a group is not necessarily the original supply set handle of the block.

N-Well and P-Well Bias

Some process technologies allow dedicated voltage supplies, instead of normal rail voltages, to be applied to n-well and p-well regions of the chip. Applying a bias voltage to a well changes the threshold voltage for transistors in the well, affecting the performance and leakage current.

The synthesis and physical implementation tools offer an optional mode to specify the n-well and p-well bias supply infrastructure using UPF commands. In this mode, the tool automatically makes supply connections to the n-well and p-well bias pins.

To enable the UPF-based well bias mode, set the `enable_bias` design attribute to `true` in the UPF command file as follows:

```
set_design_attributes -elements {.} -attribute enable_bias true
```

or in the IC Compiler II or Fusion Compiler tool:

```
set_app_options -name mv.upf.upf_bias_support -value true
```

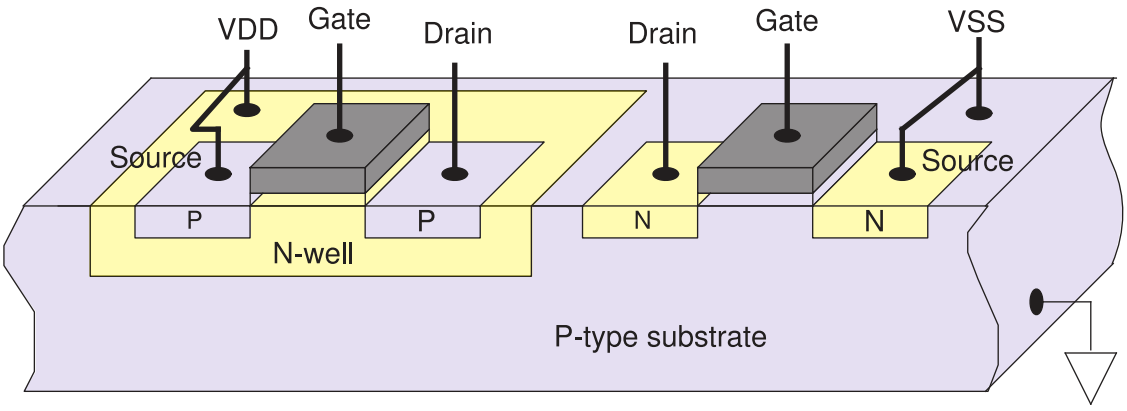
With the `enable_bias` design attribute set to `true`, supply sets and supply set handles can be used to specify the bias supply connections. The well bias pins, along with the power supply and ground pins, are connected automatically for standard cells, macros, and other types of cells.

Purpose of Well Bias

In simpler CMOS process technologies, the NMOS transistors are typically built on a p-type silicon substrate and PMOS transistors are built in n-type wells embedded in the p-type substrate, as shown in Figure 41.

Feedback

Figure 41 CMOS Technology Without Well Bias

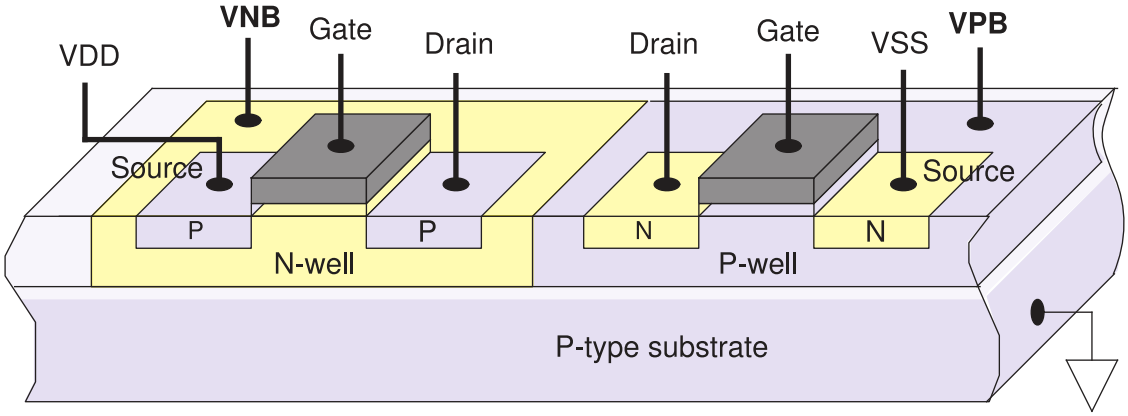


The p-type substrate is connected to the ground voltage, which prevents the p-n junctions from becoming forward biased between the p-type substrate and each of the n-type terminals of the NMOS transistor.

Similarly, the n-well is connected to the positive rail voltage, which prevents the p-n junctions from becoming forward biased between each of the p-type terminals of the PMOS transistor and the n-well. The n-well is reverse biased with respect to the p-type substrate.

Figure 42 shows how to build NMOS transistors in p-wells and PMOS transistors in n-wells, where the n-well and p-well voltages can be independently controlled. This type of construction requires the use of an advanced process technology.

Figure 42 CMOS Technology With N-Well and P-Well Bias



As in simpler CMOS technologies, the n-well can be connected to the rail voltage and the p-well can be connected to ground. However, the technology can allow different voltages to be applied to the wells to modify the behavior of the transistors, using the terminals labeled VNB (voltage n-well bias) and VPB (voltage p-well bias) in the figure.

For example, applying a voltage slightly above ground to the p-well using the VPB pin lowers the NMOS transistor threshold voltage, resulting in faster switching at the cost of higher leakage current. Conversely, applying a voltage slightly below ground on the same pin raises the NMOS transistor threshold voltage, reducing leakage current at the cost of speed.

Similarly, applying a voltage slightly above or below the rail voltage to the n-well using the VNB pin modifies the threshold voltage of the PMOS transistors to achieve better speed or less leakage.

The well voltages can be modified dynamically to increase the speed when needed or to reduce leakage during standby mode. Alternatively, the bias voltage might be applied statically to compensate for process variations or for other adjustment purposes.

Enabling the UPF Well Bias Flow

By default, the UPF-based n-well and p-well bias feature is disabled. To enable it, set the `enable_bias` design attribute to `true` at the top level of the design, preferably using a command in the UPF file:

```
set_design_attributes -elements {.} -attribute enable_bias true
```

For the IC Compiler II and Fusion Compiler tools, you must also do the following to enable UPF bias support:

```
prompt> set_app_options -name mv.upf.upf_bias_support -value true
```

This command makes the design a *bias design* and effectively designates all cell instances in the design as bias blocks.

You can selectively disable the well bias feature for lower-level blocks as follows:

```
set_design_attributes -elements {blkA blkB} -attribute enable_bias false
```

Although you can *disable* the well bias feature for a block in a design where the feature is enabled, you cannot *enable* the well bias feature for a block in a design where the feature is disabled (either explicitly or by default). In other words, to enable this feature for a block, the feature must also be enabled for all parent blocks up to the top level.

In the UPF-based well bias feature, a *bias library cell* is a logic library cell that has one or more bias pins defined as PG pins. A *nonbias library cell* is a logic library cell that does not define any bias pins. Bias and nonbias cells cannot be mixed within a given power domain. A hierarchical cell containing bias cells is called a *bias block*.

Feedback

A nonbias block inside a bias design cannot use a bias supply set from a higher level of the design. It is restricted to using its own domain-dependent supply set, just like a domain created with the `create_power_domain -supply {extra_supplies ""}` command.

Bias Blocks

A bias block has the following characteristics:

- Any supply set (including an implicit supply set) created in the scope of the bias block has up to four supply functions: `power`, `ground`, `nwell`, and `pwell`. Some bias blocks use only the `power`, `ground`, and `nwell` functions.
- Some commands, such as `create_power_domain` and `create_supply_set`, treat the additional `nwell` and `pwell` supply functions like power and ground. Other commands, such as `set_domain_supply_net`, do not support the `nwell` and `pwell` functions and therefore cannot be used in the scope of the bias block.
- The target library must contain bias library cells. Nonbias library cells cannot be used in the bias block. However, nonbias macro cells are allowed, including pad cells.
- During linking and optimization, the tool matches the voltages set on the supply functions (`power`, `ground`, `nwell`, and `pwell`) of the standard cells with the voltage map of the logic library cells.
- The connections to bias PG pins of standard cells are implicit. If you make an explicit connection to a bias PG pin of a bias cell using a supply other than the domain's default bias supply, you get a warning message with the `check_mv_design` command or a compile operation.
- During insertion of power management cells (isolation, level shifting, and retention), the tool considers the bias supplies in addition to power and ground.

The tool handles each nonbias block without considering any bias pins, even when the block exists in a bias design. During linking and optimization in a nonbias block, the tool ignores any bias pins in bias library cells.

In a nonbias design, the tool can use bias library cells because it only verifies that the `power` and `ground` supply functions of the cell match the design requirements; it ignores the bias supply pins of the cells. To ensure that only nonbias library cells are used in a nonbias design, use the `set_link_library_subset` command (or in the IC Compiler, IC Compiler II, or Fusion Compiler tool, the `set_target_library_subset` command) to restrict the selection of cells to nonbias libraries.

Feedback

Command Behavior Changes in Bias Designs

In a bias design, command behavior is modified in the following ways:

- `create_supply_set` – The supply set created by the command has two more functions, `nwell` and `pwell`, in addition to `power` and `ground`. The `-function` option specifies the supply net associated with a supply function, so the function name can be `nwell` or `pwell` as well as `power` or `ground`.
- `create_power_domain` – The command creates an implicit supply set containing the functions `nwell` and `pwell`, in addition to `power` and `ground`.
- `associate_supply_set` – The command can handle both bias and nonbias supply sets. Note that the association must be between a supply set handle and a supply set of the same type (either both bias or both nonbias).
- `add_port_state` and `add_power_state` – The commands accept negative voltage values for the `pwell` supply net.
- `set_domain_supply_net` – The command does not support well bias and cannot be used in a bias design. Use the `-supply` option of the `create_power_domain` command instead.
- `set_isolation` – If the isolation cell is inside a bias block, you cannot use the `-isolation_power_net` and `-isolation_ground_net` options in the `set_isolation` command. Use the `-isolation_supply` option instead.
- `set_retention` – If the isolation cell is inside a bias block, you cannot use the `-retention_power_net` and `-retention_ground_net` options in the `set_retention` command. Use the `-retention_supply` option instead.
- `create_power_switch` – To create a power switch for well bias, reference the `nwell` or `pwell` supply net using the `-input_supply_port` or `-output_supply_port` option.

Bias Design Rules

Each bias supply must be more always-on or equally always-on compared to the corresponding primary supply of the domain. Well bias must be achieved by changing the supply voltages connected to the well bias pins, not by changing the primary power or ground rail voltages.

Within a given power domain, any two logic elements whose bias wells physically abut must use the same bias supplies. This rule ensures electrical continuity. When you specify a supply set for an isolation or retention strategy, ensure that the bias functions of the supply set match those of the domain where the cell is placed.

Feedback

Library Modeling

In the Liberty-format description of a bias library cell, the UPF-based well bias flow allows only the following attribute definitions for bias supply pins:

- `pg_type`: `nwell` or `pwell`
- `direction`: `input`, `inout`, or `unspecified` (not `output` or `internal`)
- `physical_connection`: `routing_pin` or `device_layer`

Within each power domain, it is recommended that you consistently use library cells that have their `physical_connection` attribute set to either `routing_pin` or `device_layer`, not a mixture of both. When different physical connection parameters are defined in different libraries, you can use the `set_link_library_subset` command (or in the IC Compiler, IC Compiler II, or Fusion Compiler tool, the `set_target_library_subset` command) to restrict the selection of cells to specified libraries.

For more information about defining well bias PG pins, see the “PG Pin Syntax” section in the *Library Compiler User Guide*.

Lower-Domain Boundary

By default, each Synopsys tool considers the logical boundary of the root cells of the power domain as the boundary of the power domain. The tool can optionally consider a power domain boundary to include the boundary of another domain contained in it. This feature enables you to specify the elements on the lower-domain boundary for level-shifter and isolation strategy definitions, which gives you additional flexibility in selecting the location of the power management cells.

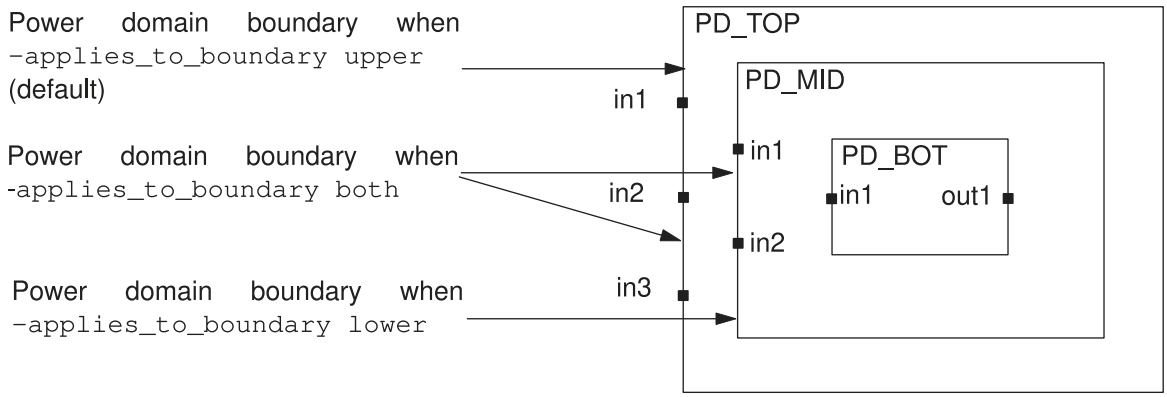
To extend the definition of the power domain boundary to the boundary of another power domain contained in it, use the `-applies_to_boundary` option of the `set_isolation`, `set_level_shifter`, or `set_repeater` commands. For example,

```
prompt> set_isolation ISO1 -domain PD_TOP -applies_to_boundary lower
```

Figure 43 illustrates the usage of the lower-domain boundary feature. In this design, the TOP block is assigned to the PD_TOP power domain, the MID block is assigned to the PD_MID power domain, and the BOT block is assigned to the PD_BOT power domain.

Feedback

Figure 43 Definition of Power Domain Boundaries Using the -applies_to_boundary Option



By default, the tool considers only in1, in2, and in3 ports of the TOP design for the PD_TOP domain boundary. This is also the case if the `-applies_to_boundary` option is set to `upper`.

When the `-applies_to_boundary` option is set to `both`, the tool considers the in1, in2, in3, MID/in1, and MID/in2 ports to be at the power domain boundary. However, the boundary does not extend to the interface of the BOT design or the PD_BOT power domain. Therefore, you can specify the MID/in1 and MID/in2 ports as valid elements in the isolation and level-shifter strategy of the power domain PD_TOP.

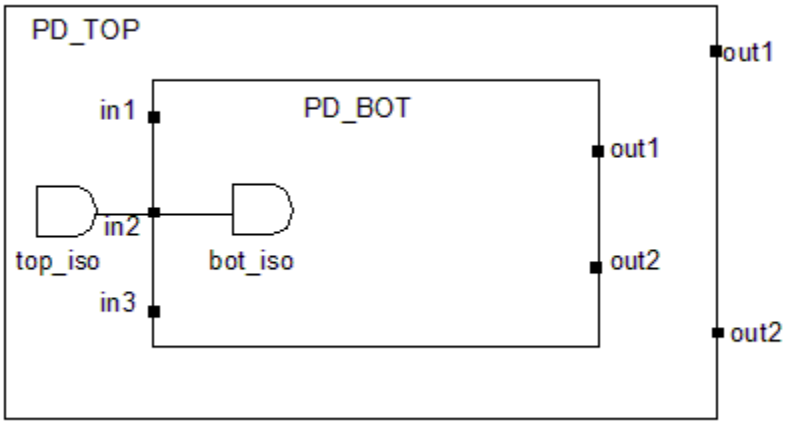
When the `-applies_to_boundary` option is set to `lower`, the tool considers the MID/in1 and MID/in2 ports to be at the power domain boundary.

With the lower-domain boundary feature, you are not required to define isolation strategies with `-location parent` in the block-level design because equivalent strategies can be defined in the parent domain.

The lower-domain boundary feature lets you specify the need or presence of isolation or level-shifter cells on both sides of the power domain boundary, with one cell inside the power domain and the other in the surrounding power domain, as shown in Figure 44.

Feedback

Figure 44 Back-to-Back Isolation Cells



For more information about the lower-domain boundary feature, see the *Power Compiler User Guide*.

Power Supply Checking, Reporting, and Collection Commands

In addition to supporting most of the UPF commands, many Synopsys tools have non-UPF commands that perform multivoltage checks that can be used at the various stages of the flow. Commands are available to report power-related objects in the design and also to create collections of such objects. The reporting and collection commands are not part of the UPF specification and vary from tool to tool, depending on the need for the commands in each tool. Some of these commands are shared by multiple tools and have similar or identical operation in different tools.

In many tools, the commands `check_mv_design` and `analyze_mv_design` perform power-specific checks that can be used to check for multivoltage violations, connection rule violations, and operating condition mismatches.

The commands `report_power_domain`, `report_power_switch`, `report_pst`, `report_power_pin_info`, `report_power_gating`, `report_supply_net`, and `report_supply_port` report the power-related objects in the design previously defined by UPF commands. Each command reports the objects existing in the design and the characteristics of the object. For example, the `report_supply_net` command lists the supply nets and shows the name, scope, power domains, supply ports, PG pins, voltage, and resolution status of each supply net.

The commands `get_power_domains`, `get_power_switches`, `get_supply_nets`, and `get_supply_ports` are commands that create collections of power-related objects in the design. You can then use a collection for custom reporting or as input to a script.

Feedback

For more information about a particular checking, reporting, or collection command, see its man page in the applicable tool.

Simple and Hierarchical Names in UPF Commands

The UPF standard requires simple names for arguments in some commands. By default, Synopsys tools do not enforce this requirement; they allow hierarchical names to be used. However, you can optionally enforce the rule for compatibility with external tools. The following commands and command arguments are affected by this rule:

```
add_power_state object_name
add_pst_state state_name
connect_supply_net supply_net_name
create_power_domain domain_name
create_power_switch switch_name
create_pst table_name
create_supply_net net_name
create_supply_port port_name
create_supply_set supply_set_name
```

To enforce the simple name requirement for execution of UPF commands in the Design Compiler or IC Compiler tool:

```
prompt> set_app_var mv_input_enforce_simple_names true
prompt> set_app_var mv_output_enforce_simple_names true
```

In this case, a UPF command that uses a hierarchical name triggers an error. To write out a UPF command containing an object with a hierarchical name, the tool also writes out `set_scope` commands to set the hierarchical scope so that it can write out the UPF command using a simple name.

The IC Compiler II and Fusion Compiler tools automatically enforce simple names. To allow hierarchical names, set the following application option:

```
prompt> set_app_options \
  -name mv.upf.input_enforce_simple_names -value false
```

UPF Command Tracking

Synthesis and physical implementation tools often perform changes that affect the UPF infrastructure of the design, such as inserting power management cells. These changes affect the UPF commands written out by the `save_upf` command. It is often desirable to see the original commands in the context of the new commands generated by the tool and written by the `save_upf` command.

Feedback

The Design Compiler, IC Compiler, IC Compiler II, and Fusion Compiler tools offer two modes for maintaining the set of UPF commands for a design:

- UPF-prime mode – The tool writes out a new UPF file, called the UPF-prime file (Design Compiler) or UPF-double-prime file (IC Compiler), which contains a mixture of old, modified, and new UPF commands.
- Golden UPF mode – The tool writes out a “supplemental” UPF file containing only the UPF commands that changed the design. The original “golden” UPF file and supplemental file together specify the power infrastructure of the design.

UPF Command Tracking in Design Compiler and IC Compiler

In the UPF-prime mode, the Design Compiler and IC Compiler tools read in a UPF script file and then later write out a modified UPF script file containing the original commands plus new UPF commands that it executes to carry out its power management tasks. The generated UPF script can then be passed on to downstream tools in the flow.

To make it easier to compare and track user-written UPF commands in the flow, the Design Compiler and IC Compiler tools maintain the original UPF commands separately from the tool-inserted UPF commands. The user-written commands and tool-inserted commands are written to separate sections in the output UPF script file. This feature is called UPF tracking. If you do not need the tracking feature and would prefer that the tool keep closely related commands together in the file, set the `mv_upf_tracking` variable to `false`. For more information, see the man page for the variable.

UPF tracking is not supported in hierarchical flows. Therefore, if you use commands such as `characterize` or `propagate_constraints -power_supply_data`, the `mv_upf_tracking` variable is automatically set to `false`, causing closely related command to be written out together in the output UPF file, and mixing the user-written and tool-inserted commands.

UPF Command Tracking in IC Compiler II and Fusion Compiler

In the IC Compiler II and Fusion Compiler tools, UPF commands that you enter during the session are automatically stored and preserved in the database. The tool keeps track of any changes it makes to your entered UPF commands. When you write out the UPF with the `save_upf` command, the tool writes out each of your entered commands as a comment line, immediately followed by the tool-modified command.

Golden UPF Flow

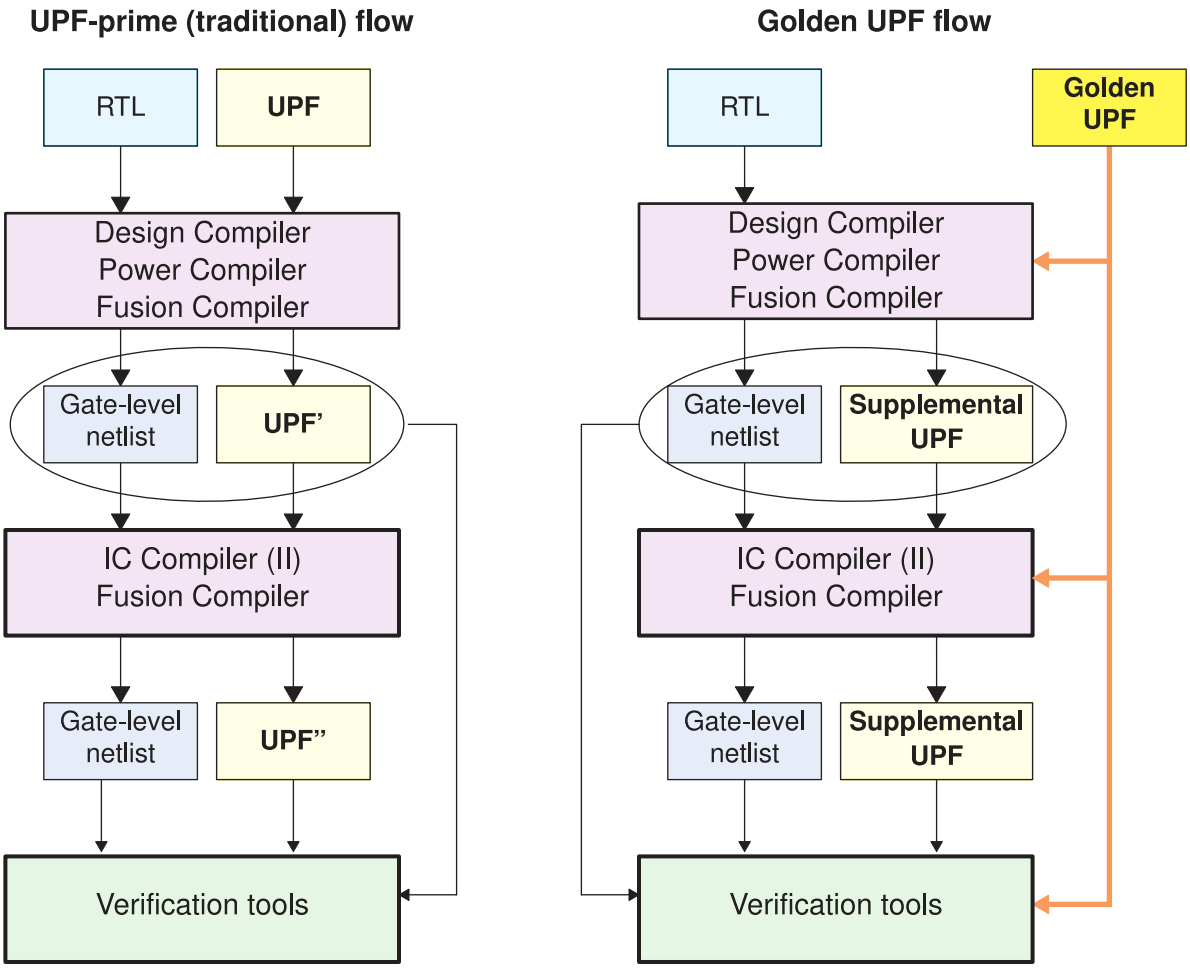
The golden UPF flow is an optional method of maintaining the UPF multivoltage power intent of the design. It uses the original “golden” UPF file throughout the synthesis,

Feedback

physical implementation, and verification steps, along with supplemental UPF files generated by the synthesis and physical implementation tools.

Figure 45 compares the traditional UPF-prime flow with the golden UPF flow.

Figure 45 UPF-Prime and Golden UPF Flows



The golden UPF flow maintains and uses the same, original “golden” UPF file throughout the flow. The synthesis and physical implementation tools write power intent changes into a separate “supplemental” UPF file. Downstream tools and verification tools use a combination of the golden UPF file and the supplemental UPF file, instead of a single UPF’ or UPF’’ file.

Feedback

The golden UPF flow offers the following advantages:

- The golden UPF file remains unchanged throughout the flow, which keeps the form, structure, comment lines, and wildcard naming used in the UPF file as originally written.
- You can use tool-specific conditional statements to perform different tasks in different tools. Such statements are lost in the traditional UPF-prime flow.
- Changes to the power intent are easily tracked in the supplemental UPF file.
- You can optionally use the Verilog netlist to store all PG connectivity information, making `connect_supply_net` commands unnecessary in the UPF files. This can significantly simplify and reduce the overall size of the UPF files.

In the Design Compiler, IC Compiler, IC Compiler II, and Fusion Compiler tools, the UPF-prime mode is enabled by default. To use the golden UPF flow, you must enable it. In the Design Compiler or IC Compiler tool:

```
prompt> set_app_var enable_golden_upf true
```

In the IC Compiler II or Fusion Compiler tool:

```
prompt> set_app_options -name mv.upf.enable_golden_upf -value true
```

After you enable this mode, to execute any UPF commands other than query commands, you must put the commands into a script and execute them with the `load_upf` command. You cannot execute them individually on the command line or with the `source` command.

For more information about using the golden UPF mode, see SolvNet article 1412864, “Golden UPF Flow Application Note.”

4

UPF Script Examples

The following multivoltage flow examples demonstrate the UPF syntax used for specifying power intent.

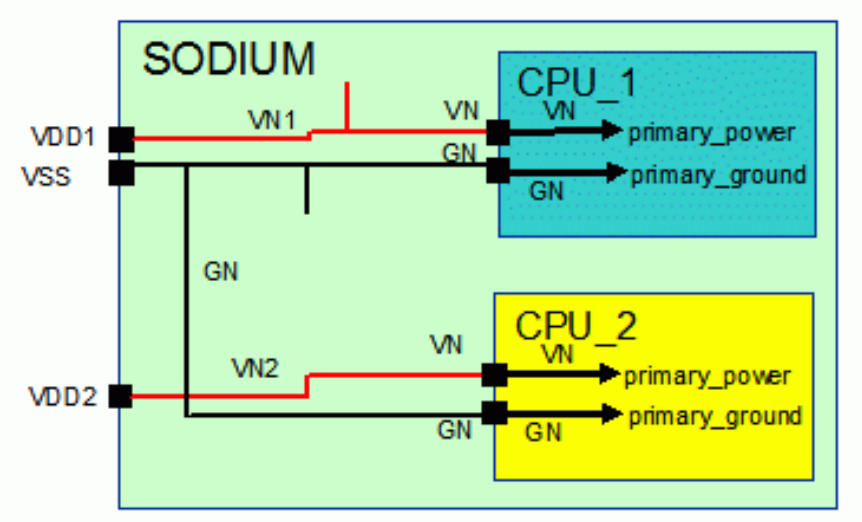
- Simple Multivoltage Design
- Supply Set Example
- Switched Power Supply Example
- Hierarchy and the get_supply_nets Command
- Power Switching States for a Macro Cell

Simple Multivoltage Design

The simple multivoltage chip design shown in Figure 46 has two power supplies, VDD1 and VDD2, at different voltage levels. The chip has two internal blocks, CPU_1 and CPU_2, both of which are instances of the same CPU block. Block CPU_1 and the top level of the chip use VDD1, whereas block CPU_2 uses VDD2. Both VDD1 and VDD2 operate as always-on power throughout the chip. The same ground, VSS, is used throughout the chip.

Feedback

Figure 46 Simple Multivoltage Chip Design



Bottom-Up Power Intent Specification

In a bottom-up flow, the lower-level CPU blocks are designed independently from the top-level chip. Therefore, their power intent must be specified at the block level and then later integrated into the chip-level power intent specification.

This is the power intent script at the block level, CPU.upf:

```
create_power_domain PD
create_supply_net VN -domain PD
create_supply_net GN -domain PD
set_domain_supply_net PD -primary_power_net VN -primary_ground_net GN
create_supply_port VN
create_supply_port GN
connect_supply_net VN -ports {VN}
connect_supply_net GN -ports {GN}
```

This is the power intent script at the top level, SODIUM.upf:

```
load_upf CPU.upf -scope CPU_1
load_upf CPU.upf -scope CPU_2
# still at scope SODIUM
create_supply_port VDD1
create_supply_port VDD2
create_supply_port VSS
create_power_domain PD
create_supply_net VN1 -domain PD
connect_supply_net VN1 -ports {VDD1 CPU_1/VN}
create_supply_net VN2 -domain PD
connect_supply_net VN2 -ports {VDD2 CPU_2/VN}
create_supply_net GN -domain PD
```

Feedback

```
connect_supply_net GN -ports {VSS CPU_1/GN CPU_2/GN}
set_domain_supply_net PD -primary_power_net VN1 -primary_ground_net GN
# PD, CPU_1/PD and CPU_2/PD are different power domains.
```

Changes Written by the save_upf Command

After the synthesis or physical implementation tool uses the original UPF power intent specification, it writes out a new UPF script with the `save_upf` command. The new script fully specifies the power intent of the design as seen from the top level. Therefore, lower-level commands originally entered at the block level are converted into commands that hierarchically specify the domains, supply nets, and supply ports.

This is the script written by the `save_upf` command at the top level:

```
create_power_domain PD -scope CPU_1
create_supply_net VN -domain CPU_1/PD
create_supply_net GN -domain CPU_1/PD
set_domain_supply_net CPU1_PD -primary_power_net CPU_1/VN \
 -primary_ground_net CPU_1/GN
create_supply_port CPU_1/VN
create_supply_port CPU_1/GN
connect_supply_net CPU_1/VN -ports {CPU_1/VN}
connect_supply_net CPU_1/GN -ports {CPU_1/GN}
 ...

create_supply_port VDD1
create_supply_port VDD2
create_supply_port VSS
create_power_domain PD
create_supply_net VN1 -domain PD
connect_supply_net VN1 -ports {VDD1 CPU_1/VN}
create_supply_net VN2 -domain PD
connect_supply_net VN2 -ports {VDD2 CPU_2/VN}
create_supply_net GN -domain PD
connect_supply_net GN -ports {VSS CPU_1/GN CPU_2/GN}
set_domain_supply_net PD -primary_power_net VN1 -primary_ground_net GN
```

Top-Down Power Intent Specification

If the chip is designed from the top down, the power intent of the design can be specified directly from the top level, if desired, as demonstrated by the following example:

```
create_power_domain PD_CPU_1 -elements {CPU_1}
create_power_domain PD_CPU_2 -elements {CPU_2}
create_power_domain PD_SODIUM
create_supply_port VDD1
create_supply_port VDD2
create_supply_port VSS
create_supply_net VN1 -domain PD_CPU_1
create_supply_net VN1 -domain PD_SODIUM -reuse
```

```
connect_supply_net VN1 -ports {VDD1}
create_supply_net VN2 -domain PD_CPU_2
connect_supply_net VN2 -ports {VDD2}
create_supply_net GN -domain PD_CPU_1
create_supply_net GN -domain PD_CPU_2 -reuse
create_supply_net GN -domain PD_SODIUM -reuse
connect_supply_net GN -ports {VSS}
set_domain_supply_net PD_CPU_1 \
 -primary_power_net VN1 -primary_ground_net GN
set_domain_supply_net PD_CPU_2 \
 -primary_power_net VN2 -primary_ground_net GN
set_domain_supply_net PD_SODIUM \
 -primary_power_net VN1 -primary_ground_net GN
```

Supply Set Example

The following script demonstrates the usage of explicit supply sets.

```
# UPF section for Explicit Supply Set definition
## Power Domain Supply Sets

#set_scope /
create_supply_set VDD_VSS
create_supply_set VDD_LOW_VSS
create_supply_set VDDS_VSS_p0
create_supply_set VDDS_VSS_p1
create_supply_set VDDS_VSS_misc
create_supply_set VDD_LOWS_VSS_p2
create_supply_set VDD_LOWS_VSS_p3

# UPF section for Power Domain definition
## Number of PDs: 6
create_power_domain TOP \
   -supply {extra_supplies_0 VDD_VSS} \
   -supply {extra_supplies_1 VDD_LOW_VSS}
set_domain_supply_net TOP -primary_power_net VDD_VSS.power \
   -primary_ground_net VDD_VSS.ground
create_power_domain LEON3_p0 \
  -elements {u0_0/pd_switchable} \
  -supply {extra_supplies_0 VDDS_VSS_p0} \
  -supply {extra_supplies_1 VDD_VSS}
set_domain_supply_net LEON3_p0 -primary_power_net VDDS_VSS_p0.power \
  -primary_ground_net VDDS_VSS_p0.ground
create_power_domain LEON3_p1 \
  -elements {u0_1/pd_switchable} \
  -supply {extra_supplies_0 VDDS_VSS_p1} \
  -supply {extra_supplies_1 VDD_VSS}
set_domain_supply_net LEON3_p1 -primary_power_net VDDS_VSS_p1.power \
  -primary_ground_net VDDS_VSS_p1.ground
create_power_domain LEON3_p2 \
  -elements {u0_2/pd_switchable} \
```

```
  -supply {extra_supplies_0 VDD_LOWS_VSS_p2} \
  -supply {extra_supplies_1 VDD_VSS} \
  -supply {extra_supplies_2 VDD_LOW_VSS}
set_domain_supply_net LEON3_p2 -primary_power_net VDD_LOWS_VSS_p2.power \
-primary_ground_net VDD_LOWS_VSS_p2.ground
create_power_domain LEON3_p3 \
  -elements {u0_3/pd_switchable} \
  -supply {extra_supplies_0 VDD_LOWS_VSS_p3} \
  -supply {extra_supplies_1 VDD_VSS} \
  -supply {extra_supplies_2 VDD_LOW_VSS}
set_domain_supply_net LEON3_p3 -primary_power_net VDD_LOWS_VSS_p3.power \
  -primary_ground_net VDD_LOWS_VSS_p3.ground
create_power_domain LEON3_misc \
  -elements {u_m/u_m_pd} \
  -supply {extra_supplies_0 VDDS_VSS_misc} \
  -supply {extra_supplies_1 VDD_VSS}
set_domain_supply_net LEON3_misc -primary_power_net VDDS_VSS_misc.power
  -primary_ground_net VDDS_VSS_misc.ground
```

Port and Supply Set Connections

```
# UPF section for port and supply set connections
## High Voltage supply 1.08
create_supply_port VDD
connect_supply_net VDD_VSS.power -ports VDD
## Low Voltage Supply 0.7
create_supply_port VDD_LOW
connect_supply_net VDD_LOW_VSS.power -ports VDD_LOW
## Ground nets
create_supply_port VSS
connect_supply_net VDD_VSS.ground -ports VSS

# Power Switches
create_power_switch leon3_p0_sw \
  -domain LEON3_p0 \
  -input_supply_port {in VDD_VSS.power} \
  -output_supply_port {out VDDS_VSS_p0.power} \
  -control_port {p0_sd u_m/u_power_controller_top/p0_sd} \
  -on_state {p0_on_state in {!p0_sd}}
create_power_switch leon3_p1_sw \
  -domain LEON3_p1 \
  -input_supply_port {in VDD_VSS.power} \
  -output_supply_port {out VDDS_VSS_p1.power} \
  -control_port {p1_sd u_m/u_power_controller_top/p1_sd} \
  -on_state {p1_on_state in {!p1_sd}}
create_power_switch leon3_p2_sw \
  -domain LEON3_p2 \
  -input_supply_port {in VDD_LOW_VSS.power} \
  -output_supply_port {out VDD_LOWS_VSS_p2.power} \
  -control_port {p2_sd u_m/u_power_controller_top/p2_sd} \
  -on_state {p2_on_state in {!p2_sd}}
```

Feedback

```
create_power_switch leon3_p3_sw \
  -domain LEON3_p3 \
  -input_supply_port {in VDD_LOW_VSS.power} \
  -output_supply_port {out VDD_LOWS_VSS_p3.power} \
  -control_port {p3_sd u_m/u_power_controller_top/p3_sd} \
  -on_state {p3_on_state in {!p3_sd}}
create_power_switch leon3_misc_sw \
  -domain LEON3_misc \
  -input_supply_port {in VDD_VSS.power} \
  -output_supply_port {out VDDS_VSS_misc.power} \
  -control_port {all_sd u_m/u_power_controller_top/all_sd} \
  -on_state {misc_on_state in {!all_sd}}

# Updates
create_supply_set VDD_LOW_VSS  -function {ground VDD_VSS.ground} -update
create_supply_set VDDS_VSS_p0  -function {ground VDD_VSS.ground} -update
create_supply_set VDDS_VSS_p1  -function {ground VDD_VSS.ground} -update
create_supply_set VDDS_VSS_misc -function {ground VDD_VSS.ground} -update
create_supply_set VDD_LOWS_VSS_p2 -function {ground VDD_VSS.ground} \
  -update
create_supply_set VDD_LOWS_VSS_p3 -function {ground VDD_VSS.ground} \
  -update
```

Power States

```
# UPF Power States
## Top Supply States
add_power_state VDD_VSS -state TOP_HV \
  {-supply_expr {power == `{FULL_ON, 1.08}}}
add_power_state VDD_VSS -state GND \
  {-supply_expr {ground == `{FULL_ON, 0.0}}}

add_power_state VDD_LOW_VSS -state TOP_LV \
  {-supply_expr {power == `{FULL_ON, 0.7}}}

## Off Supply states
add_power_state VDDS_VSS_p0 -state P0_HV \
  {-supply_expr {power == `{FULL_ON, 1.08}}}
add_power_state VDDS_VSS_p0 -state P0_OFF \
  {-supply_expr {power == `{OFF}}}

add_power_state VDDS_VSS_p1 -state P1_HV \
  {-supply_expr {power == `{FULL_ON, 1.08}}}
add_power_state VDDS_VSS_p1 -state P1_OFF \
  {-supply_expr {power == `{OFF}}}

add_power_state VDD_LOWS_VSS_p2 -state P2_LV \
  {-supply_expr {power == `{FULL_ON, 0.7}}}
add_power_state VDD_LOWS_VSS_p2 -state P2_OFF \
  {-supply_expr {power == `{OFF}}}
```

Feedback

```
add_power_state VDD_LOWS_VSS_p3 -state P3_LV \
  {-supply_expr {power == `{FULL_ON, 0.7}}}
add_power_state VDD_LOWS_VSS_p3 -state P3_OFF \
  {-supply_expr {power == `{OFF}}}

add_power_state VDDS_VSS_misc -state MISC_HV\
  {-supply_expr {power == `{FULL_ON,1.08}}}
add_power_state VDDS_VSS_misc -state MISC_OFF\
  {-supply_expr {power == `{OFF}}}

# PST
create_pst LEON3_MP_PST -supplies \
{VDD_VSS.power VDD_LOW_VSS.power VDDS_VSS_p0.power VDDS_VSS_p1.power \
VDD_LOWS_VSS_p2.power VDD_LOWS_VSS_p3.power VDDS_VSS_misc.power \
VDD_VSS.ground}

add_pst_state INIT      -pst LEON3_MP_PST -state {TOP_HV TOP_LV  \
                        P0_OFF  P1_OFF  P2_OFF  P3_OFF  MISC_OFF  GND}
add_pst_state P0_BOOT   -pst LEON3_MP_PST -state {TOP_HV TOP_LV  \
                        P0_HV   P1_OFF  P2_OFF  P3_OFF  MISC_OFF  GND}
add_pst_state P0_P1_BOOT -pst LEON3_MP_PST -state {TOP_HV TOP_LV  \
                        P0_HV   P1_HV   P2_OFF  P3_OFF  MISC_OFF  GND}
add_pst_state P0_P1_P3_BOOT -pst LEON3_MP_PST -state {TOP_HV TOP_LV  \
                        P0_HV   P1_HV   P2_LV   P3_OFF  MISC_OFF  GND}
add_pst_state ALL_ON    -pst LEON3_MP_PST -state {TOP_HV TOP_LV  \
                        P0_HV   P1_HV   P2_LV   P3_LV   MISC_HV   GND}
add_pst_state HIGH_PERF -pst LEON3_MP_PST -state {TOP_HV TOP_LV  \
                        P0_HV   P1_HV   P2_OFF  P3_OFF  MISC_HV   GND}
add_pst_state LOW_PERF  -pst LEON3_MP_PST -state {TOP_HV TOP_LV  \
                        P0_OFF  P1_OFF  P2_LV   P3_LV   MISC_HV   GND}
add_pst_state P3_DROWSY -pst LEON3_MP_PST -state {TOP_HV TOP_LV  \
                        P0_OFF  P1_OFF  P2_OFF  P3_LV   MISC_HV   GND}
add_pst_state P2_DROWSY -pst LEON3_MP_PST -state {TOP_HV TOP_LV  \
                        P0_OFF  P1_OFF  P2_LV   P3_OFF  MISC_HV   GND}
add_pst_state P1_DROWSY -pst LEON3_MP_PST -state {TOP_HV TOP_LV  \
                        P0_HV   P1_OFF  P2_OFF  P3_LV   MISC_HV   GND}
add_pst_state P0_DROWSY -pst LEON3_MP_PST -state {TOP_HV TOP_LV  \
                        P0_OFF  P1_HV   P2_OFF  P3_LV   MISC_HV   GND}
add_pst_state ULTRA_DROWSY  -pst LEON3_MP_PST -state {TOP_HV TOP_LV  \
                        P0_OFF  P1_HV   P2_OFF  P3_OFF  MISC_OFF  GND}
```

Multivoltage Strategies

```
# UPF section for MV strategies definition
## Number of isolation strategies: 5
## Number of retention strategies: 1
## Number of power switches: 5

### PD LEON3_p0
set_isolation leon3_p0_iso_out \
  -domain LEON3_p0 \
```

Feedback

```
  -isolation_power_net VDD_VSS.power \
  -isolation_ground_net VDD_VSS.ground \
  -clamp_value 1 \
  -applies_to outputs

set_isolation_control leon3_p0_iso_out \
  -domain LEON3_p0 \
  -isolation_signal u_m/u_power_controller_top/p0_isolation \
  -isolation_sense high \
  -location self

### PD LEON3_p1

set_isolation leon3_p1_iso_out \
  -domain LEON3_p1 \
  -isolation_power_net VDD_VSS.power \
  -isolation_ground_net VDD_VSS.ground \
  -clamp_value 1 \
  -applies_to outputs

set_isolation_control leon3_p1_iso_out \
  -domain LEON3_p1 \
  -isolation_signal u_m/u_power_controller_top/p1_isolation \
  -isolation_sense high \
  -location self

### PD LEON3_p2
set_isolation leon3_p2_iso_out \
  -domain LEON3_p2 \
  -isolation_power_net VDD_LOW_VSS.power \
  -isolation_ground_net VDD_LOW_VSS.ground \
  -clamp_value 1 \
  -applies_to outputs

set_isolation_control leon3_p2_iso_out \
  -domain LEON3_p2 \
  -isolation_signal u_m/u_power_controller_top/p2_isolation \
  -isolation_sense high \
  -location self
```

Level Shifter Strategy

```
# set the level shifter strategy according to the library capability
set_level_shifter -domain LEON3_p2 -applies_to inputs \
  -location self u0_2_ls_in
set_level_shifter -domain LEON3_p2 -applies_to outputs
  -location parent u0_2_ls_out

### PD LEON3_p3

set_isolation leon3_p3_iso_out \
```

Feedback

```
  -domain LEON3_p3 \
  -isolation_power_net VDD_LOW_VSS.power \
  -isolation_ground_net VDD_LOW_VSS.ground \
  -clamp_value 1 \
  -applies_to outputs

set_isolation_control leon3_p3_iso_out \
  -domain LEON3_p3 \
  -isolation_signal u_m/u_power_controller_top/p3_isolation \
  -isolation_sense high \
  -location self

# set the level shifter strategy according to the library capability
set_level_shifter -domain LEON3_p3 -applies_to inputs \
  -location self u0_3_ls_in
set_level_shifter -domain LEON3_p3 -applies_to outputs\
  -location parent u0_3_ls_out

### PD LEON3_misc
set_isolation leon3_misc_iso_out \
  -domain LEON3_misc \
  -isolation_power_net VDD_VSS.power \
  -isolation_ground_net VDD_VSS.ground \
  -clamp_value 1 \
  -applies_to outputs

set_isolation_control leon3_misc_iso_out \
  -domain LEON3_misc \
  -isolation_signal u_m/u_power_controller_top/all_isolation \
  -isolation_sense high \
  -location self
set_isolation leon3_misc_no_pci_req -domain LEON3_misc \
   =-elements { u_m/u_m_pd/pcio[REQEN] } -no_isolation

set_retention misc_ret -domain LEON3_misc -retention_power_net \
  VDD_VSS.power -retention_ground_net  VDD_VSS.ground
set_retention_control misc_ret -domain LEON3_misc \
        -save_signal {u_m/u_power_controller_top/all_save high} \
        -restore_signal {u_m/u_power_controller_top/all_restore low}
map_retention_cell -domain LEON3_misc -lib_cells { \
RDFFSRX1         RDFFSRASRX1 \
RSDFFSRX1        RSDFFSRASRX1 \
RDFFSRX2         RDFFSRASRX2 \
RSDFFSRX2        RSDFFSRASRX2 \
RDFFSRX1_HVT     RDFFSRASRX1_HVT  \
RSDFFSRX1_HVT    RSDFFSRASRX1_HVT  \
RDFFSRX2_HVT     RDFFSRASRX2_HVT  \
RSDFFSRX2_HVT    RSDFFSRASRX2_HVT  \
RDFFSRX1_LVT     RDFFSRASRX1_LVT  \
RSDFFSRX1_LVT    RSDFFSRASRX1_LVT  \
RDFFSRX2_LVT     RDFFSRASRX2_LVT  \
RSDFFSRX2_LVT    RSDFFSRASRX2_LVT  \
} misc_ret
```

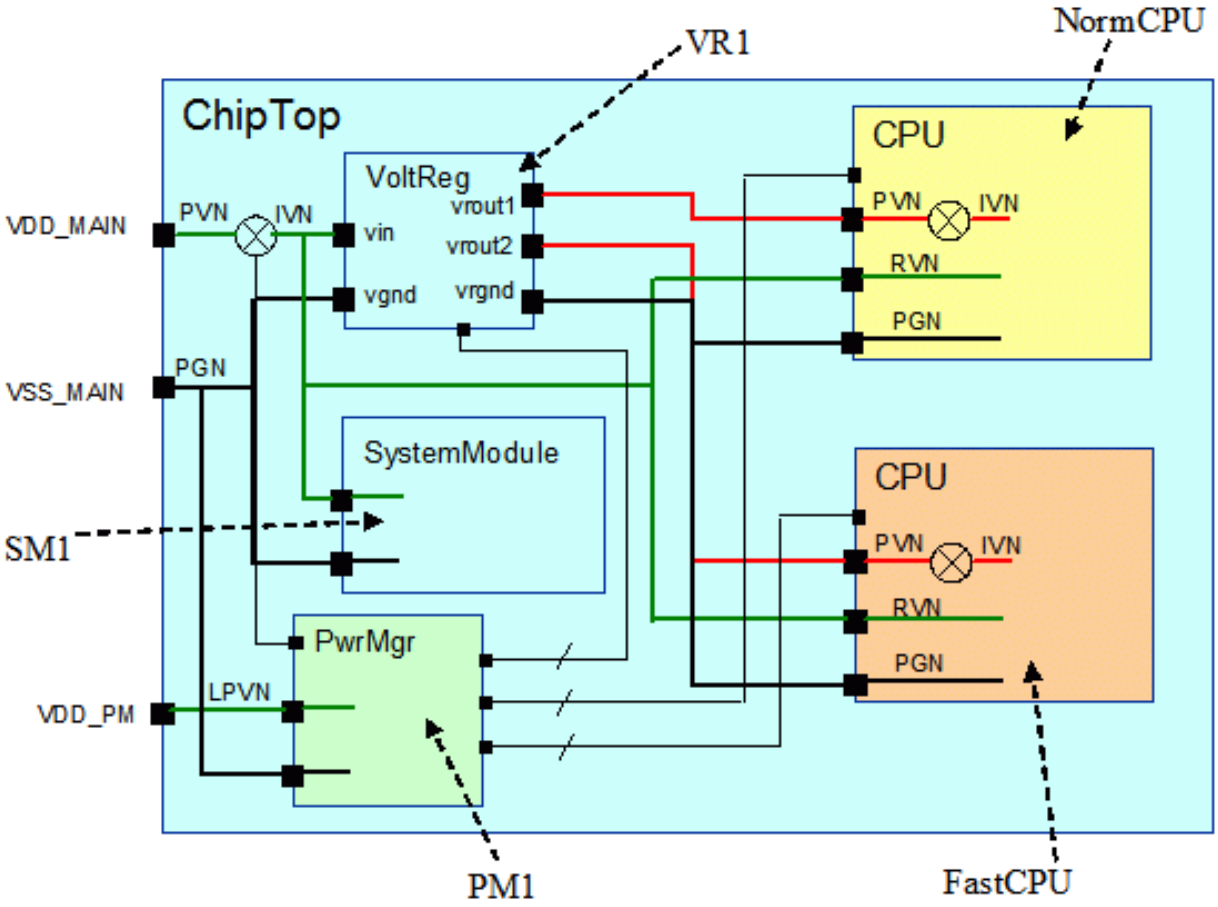
Feedback

```
set_port_attributes -elements {.} \
  -attribute related_supply_default_primary true
```

Switched Power Supply Example

The chip design shown in Figure 47 has two external power supplies, VDD_MAIN and VDD_PM. The chip has two internal blocks, CPU_1 and CPU_2, both of which are instances of the same CPU block. An on-chip voltage regulator produces two supplies, vrout1 and vrout2, at different voltage levels. The supply vrout1 powers the “normal” CPU block and the supply vrout2, at a higher voltage level, powers the “fast” CPU block. The VDD_MAIN power supply is switchable on the chip. Each CPU block has its own internal switch to turn on and turn off its own power supply. A power manager logic block generates the signals that control the power switches.

Figure 47 Simple Hierarchical Chip Design



Feedback

The RTL code for this chip design has the following form:

```
module CPU(...);
...
endmodule

module SystemModule(...);
...
endmodule

module chiptop(...);
CPU FastCPU(...);
CPU NormCPU(...);
PwrMgr PM1(...);
SystemModule SM1(...);
VoltReg VR1(...);
endmodule
```

The voltage regulator block, I VoltReg, is a user-instantiated macro cell, with an input at 1.2 volts, producing two power supply outputs at varying voltage levels:

- vr1: 0.7V, 0.8V, 0.9V, for NormCPU
- vr2: 1.0V, 1.1V, 1.2V, for FastCPU

The voltage regulator is controlled by PwrMgr to select different states and produces some acknowledge signals to the PwrMgr.

This is the lower-level power intent specification script for the CPU block, cpu.upf:

```
create_power_domain PD1

create_supply_net PVN -domain PD1
create_supply_port PVN -domain PD1
connect_supply_net PVN -ports PVN
create_supply_net IVN -domain PD1
create_power_switch sw1 \
  -domain PD1 \
  -input_supply_port {vin PVN} \
  -output_supply_port {vout IVN} \
  -control_port {ctrl sw_ctrl_net} \
  -on_state {state1 vin {ctrl}}
create_supply_net PGN -domain PD1
create_supply_port PGN -domain PD1
connect_supply_net PGN -ports PGN
create_supply_net RVN -domain PD1
create_supply_port RVN -domain PD1
connect_supply_net RVN -ports RVN

set_domain_supply_net PD1  \
  -primary_power_net IVN -primary_ground_net PGN

set_retention retent1 \
```

```
  -domain PD1 \
  -retention_power_net RVN -retention_ground_net PGN
set_retention_control retent1 \
  -domain PD1 \
  -save_signal {cpu_state_save high} \
  -restore_signal {cpu_state_restore high}

set_isolation iso1 \
  -domain PD1 \
  -isolation_power_net RVN -isolation_ground_net PGN \
  -clamp_value 1 \
  -applies_to outputs
set_isolation_control iso1 \
  -domain PD1 \
  -isolation_signal cpu_iso \
  -isolation_sense low \
  -location self
```

This is the top-level power intent specification, chiptop.upf:

```
set_scope FastCPU
load_upf cpu.upf
set_scope ../NormCPU
load_upf cpu.upf
set_scope
# set_scope makes chiptop not reusable

# PD for the PwrMgr, powered separately
create_power_domain PD2 \
  -elements {PM1}
create_supply_net LPVN -domain PD2
create_supply_port VDD_PM -domain PD2

# PD for glue logic and SystemModule
create_power_domain PD1
create_supply_net PVN -domain PD1
create_supply_port VDD_MAIN -domain PD1
connect_supply_net PVN -ports {VDD_MAIN}
create_supply_net IVN -domain PD1
create_power_switch sw1 \
  -domain PD1 \
  -input_supply_port {vin PVN} \
  -output_supply_port {vout IVN} \
  -control_port {ctrl sw_ctrl_net} \
  -on_state {on_state vin {ctrl}}
create_supply_net PGN -domain PD1
create_supply_net PGN -domain PD2 -reuse
create_supply_port VSS_MAIN -domain PD1

# Explicit connections of pre-regulated supply nets
connect_supply_net PGN -ports {VSS_MAIN VR1/vgnd}
connect_supply_net IVN -ports {VR1/vin}
connect_supply_net LPVN -ports {VDD_PM}
```

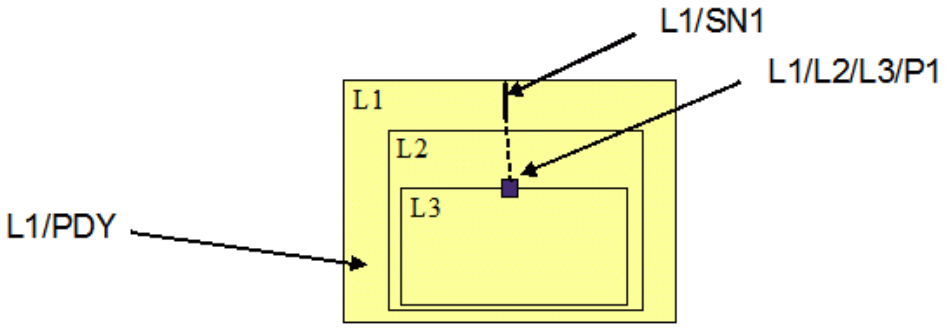
Feedback

```
# Implicit connections of pre-regulated supply nets
set_domain_supply_net PD1 -primary_power_net IVN -primary_ground_net PGN
set_domain_supply_net PD2 -primary_power_net LPVN -primary_ground_net PGN

set_isolation \
  -domain PD1 \
  -isolation_power_net PVN

set_isolation_control -domain PD1 \
  -isolation_signal iso_sig_net

# Explicit connections to soft IPs
create_supply_net RegVN1 -domain PD2
create_supply_net RegVN2 -domain PD2
create_supply_net RegVG -domain PD2
connect_supply_net RegVN1 -ports {VR1/vrout1 NormCPU/PVN}
connect_supply_net RegVN2 -ports {VR1/vrout2 FastCPU/PVN}
connect_supply_net IVN -ports {NormCPU/RVN FastCPU/RVN}
connect_supply_net RegVG -ports {VR1/vrgnd NormCPU/PGN FastCPU/PGN}
```

Hierarchy and the get_supply_nets Command

The `get_supply_nets` command finds the logical net name associated with a supply net in the specified domain for the specified scope. Figure 48 and the following code example demonstrate how to use the `get_supply_nets` command to make a supply net connection across hierarchical levels.

Figure 48 Hierarchical Supply Connection Using get_supply_nets



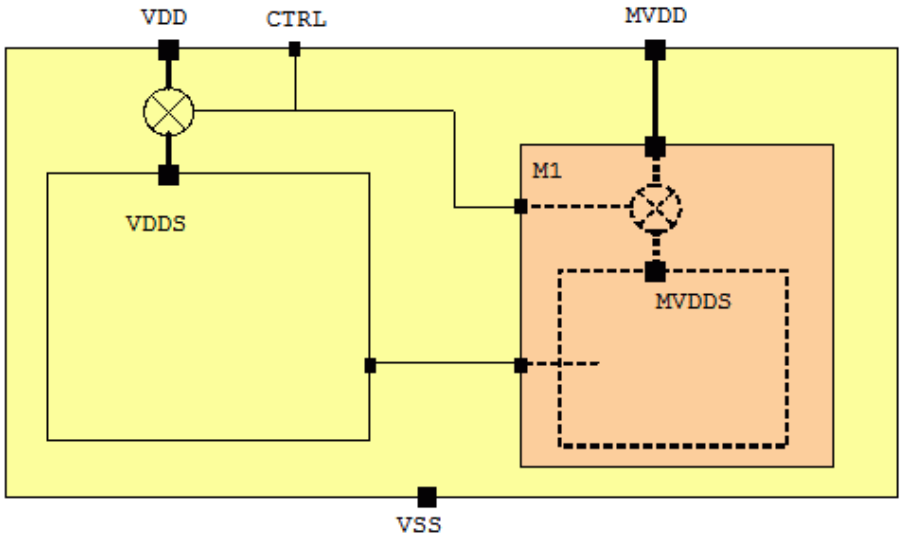

```
create_power_domain PDY -scope L1 -elements {L1}
create_supply_net SN1 -domain L1/PDY
create_supply_port L1/L2/L3/P1
set n [get_supply_nets SN1 -domain L1/PDY -scope L1/L2]
connect_supply_net $n -ports {L1/L2/L3/P1}
# an alternative to connect_supply_net L1/SN1 -ports {L1/L2/L3/P1}
```

Feedback

Power Switching States for a Macro Cell

In the power intent example shown in Figure 49, the macro cell has an internal power switch. The rest of the circuit within the macro cell gets its power from the output of the internal switch. The internal power switch is controlled by a signal pin at the cell interface. The signal driving this macro control pin is the same as the signal controlling the power switch driving the rest of the design outside the macro. Furthermore, VDD and MVDD are always on, but with different voltages. In other words, the on/off states of supply net VDDS and MVDDS (which are not accessible from the design) are synchronized together, although they are at different voltages.

Figure 49 Macro Cell With Power Switch

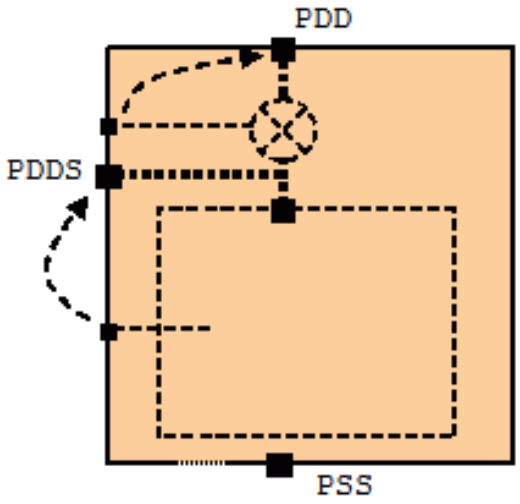


The design-level power switch is not yet available in the RTL. However, a UPF script is needed to create this power switch and describe the supply scheme of the design, such that all tools recognize that no isolation cell is needed between the macro cell and the rest of the circuit in the design.

The macro cell vendor must provide a Liberty cell model for the macro so that all signal pins are related to the switched power pin of the macro. The macro cell model should look like Figure 50 schematically.

Feedback

Figure 50 Macro Cell Power Connections



The dashed arrows show the “related_power_pin” relationship between a signal pin and a power pin on the macro. PDDS should have an “internal” pin type so that it is not automatically connected to the domain supplies.

Given the availability of a correct liberty cell model, the UPF file can be written as follows:

```
create_power_domain PDT
create_power_domain PDM_-elements {M1}
create_supply_port VDD
create_supply_port MVDD
create_supply_port VSS
create_supply_net VDD -domain PDT
create_supply_net MVDD -domain PDM
create_supply_net VSS -domain PDT
create_supply_net VSS -domain PDM -reuse
# power switch for domain PDT
create_power_switch SW1 -domain PDT \
  -output_supply_port {sout VDDS} \
  -input_supply_port {sin VDD} \
  -control_port {sctrl ctrl} \
  -on_state {on1 sin {ctrl}}
# connection for domain PDT
connect_supply_net VDD -ports {VDD}
connect_supply_net VSS -ports {VSS}
set_domain_supply_net PDT \
  -primary_power_net VDDS -primary_ground_net VSS
# connection for domain PDM
connect_supply_net MVDD -ports {MVDD}
set_domain_supply_net PDM \
  -primary_power_net MVDD -primary_ground_net VSS

# describe the power states so that checker does not complain
add_port_state SW1/sout -state {s1 0.7} -state {s2 0.8} -state {s3 off}
```

Feedback

```
add_port_state M1/PDDS -state {m1 1.0} -state {m2 0.9} -state {m3 off}
create_pst pst1 -supplies {SW1/sout M1/PDDS}
add_pst_state state1 -pst pst1 -state {0.7 0.9}
add_pst_state state2 -pst pst1 -state {0.8 0.9}
add_pst_state state3 -pst pst1 -state {0.8 1.0}
add_pst_state state4 -pst pst1 -state {off off}
# explicit directive to say no isolation
set_isolation iso1 -domain PDM -no_isolation
```

5

Tool-Specific Usage Recommendations

This chapter provides information about using Synopsys tools for low-power design and analysis. It contains the following sections:

Note:

The information in this book applies to the most recent releases of Synopsys products. Due to recent changes in service pack releases, current product features might be different from what is described in this book. Refer to the individual product release notes for the most current information.

Multivoltage Verification Using VCS NLP and VC LP

In the Synopsys functional verification flow, you can use the VCS Native Low Power (NLP) tool for multivoltage functional simulation and the VC LP tool for static multivoltage rule checking. The VCS NLP tool simulates the design to verify the impact of voltage changes in a power-managed chip, allowing you to detect any low power design issues accurately and reliably. The VC LP tool checks multivoltage rules and reports any problems related to power architecture, power intent consistency, and power connectivity.

Feedback

VCS NLP Native Low Power Simulation

A VCS NLP simulation verifies the power management architecture, including the power-up and power-down sequence and the policies for retention, isolation, and level shifting.

This section describes the VCS NLP low power simulation in the following topics:

- Getting Started with Power-Aware Simulation
- Debugging Low-Power Designs Using Verdi
- Coverage Support for Low-Power Objects
- Power-Aware Gate-Level Netlist Simulation Flow

Getting Started with Power-Aware Simulation

VCS NLP provides a basic low power simulation capability in a UPF-based flow using power-gating and retention techniques.

VCS NLP equips VCS to natively perform voltage-level aware simulation with a complete understanding of the UPF and RTL prior to the implementation flows. This comprehensively verifies correct behavior of designs that use advanced voltage control techniques for power management, and catches potential low power bugs early in the design process.

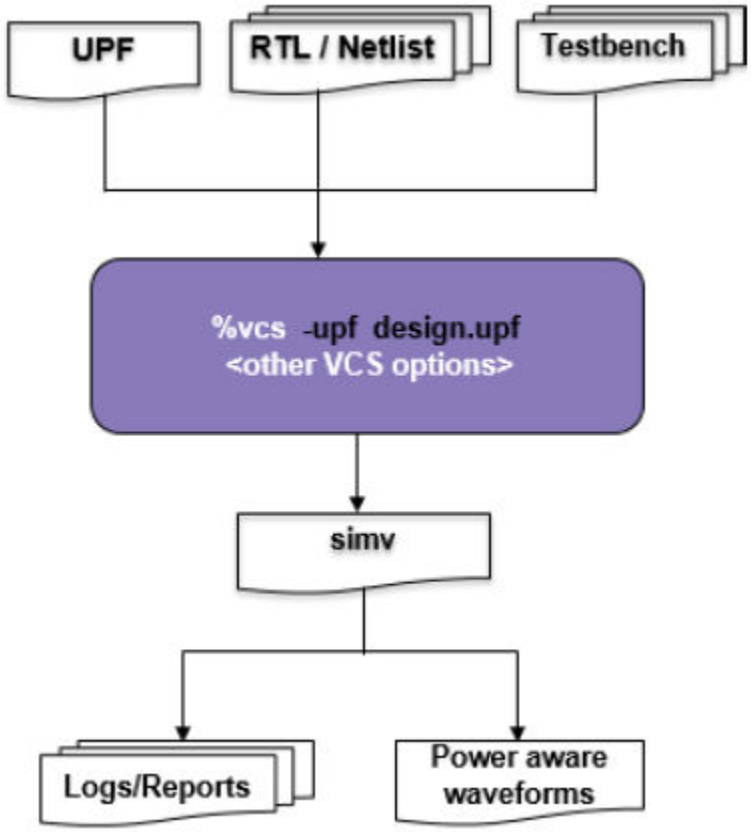
VCS NLP works with normal RTL, so legacy RTL blocks are easily reused without modifying the RTL code. New reusable blocks can be created independently of the power-aware environment. VCS NLP accepts the same testbench that is used in the standard flows (optionally augmented for low power checks).

Feedback

VCS NLP Low Power Simulation Flow

Specifying the `-upf`option at compile time in the VCS tool automatically enables VCS NLP. You must specify the UPF file using the `-upf` option at `vcs` step as shown in Figure 51:

Figure 51 VCS NLP Low Power Simulation Flow



To use low power mixed designs in VCS MX:

1. Analyze the Verilog and VHDL files using `vlogan` or `vhdlan`.

   For VHDL files,

   ```
   % vhdlan [vhdlan_options] file1.vhd file2.vhd
   ```

   For Verilog files,

   ```
   % vlogan [vlogan_options] file1.v file2.v
   ```

   For SystemVerilog files,

   ```
   % vlogan -sverilog [vlogan_options] file1.sv file2.sv file3.v
   ```

2. Power intent specified in UPF is loaded into VCS along with the design and the testbench using the `-upf` compile option. The design top is also set for the module for which the UPF file is written either using the `set_design_top` UPF command or the `-power_top` compile time option:

   ```
   % vcs <other options> -upf <file name> -power_top <module name>
   ```

3. During simulation, you can specify the runtime configuration file to set the runtime design attributes and unified message control options.

   ```
   % simv -power power_config.tcl
   ```

   Here is an example of the `power_config.tcl` file:

   ```
   set_design_attributes -elements PD_Core -attribute \
   SNPS_random_corruption 01X
   ```

Debugging Low-Power Designs Using Verdi

VCS NLP has the native Verdi integrated, which can be used for debugging low power simulations. Verdi supports IEEE 1801 UPF and allows you to visualize, trace, and analyze the source of low power events. It simplifies low power debugging by allowing you to visualize the power intent (UPF) and supports features to determine whether an unexpected design behavior is caused by the functional logic or a power event.

For VCS and Verdi databases to be consistent for accurate debugging, VCS NLP instruments the low power objects in HDL design to mimic the power intent specified in the UPF. Verdi allows you to debug low power objects instrumented by VCS NLP.

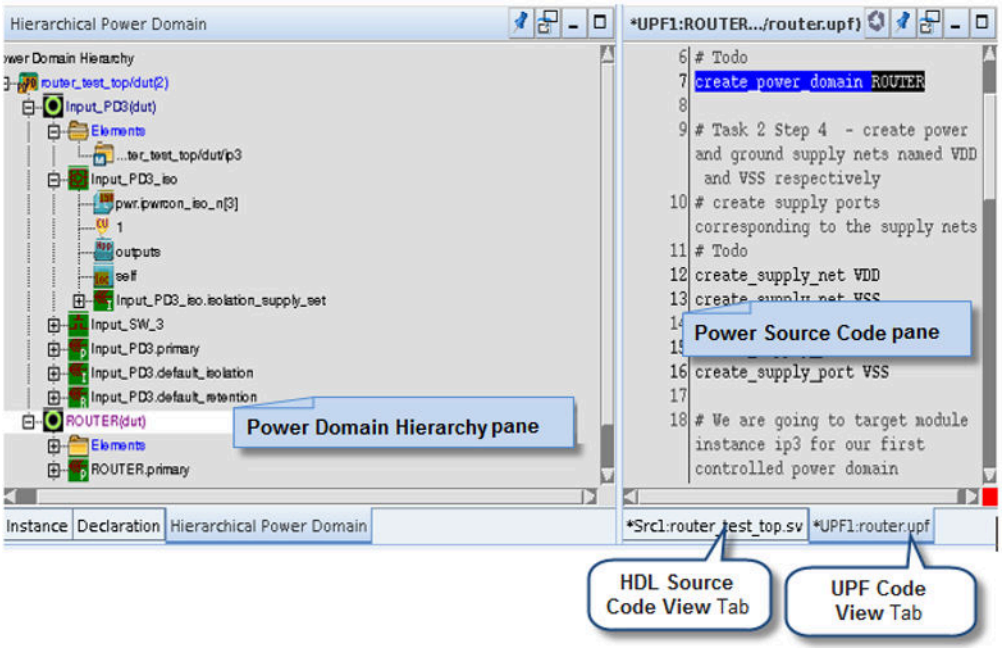
The following is the syntax to compile a low power design:

```
% vcs -sverilog design_file -upf upf_file -power_top design_top -kdb \
 -debug_access+all compile_options
```

The following is the syntax to invoke the Verdi GUI:

```
% simv -verdi
```

Figure 52 Power-Aware Debug



Coverage Support for Low-Power Objects

You can use the `-power=coverage` compile-time option to automatically create covergroups for the low power objects based on the power intent. For more information on covergroups, refer to *IEEE Std 1800-2012, IEEE Standard for SystemVerilog*

The following is the syntax to enable coverage on a low power design:

```
% vcs -sverilog <design_file> <other_compile_options> -upf <upf_file>
-debug_access+all -power=coverage
```

The following is the syntax to load a coverage database:

```
% verdi -cov -covdir simv.vdb &
```

Where `simv.vdb` is the coverage database generated by VCS after simulating the design.

The covergroups are created for the following low power objects:

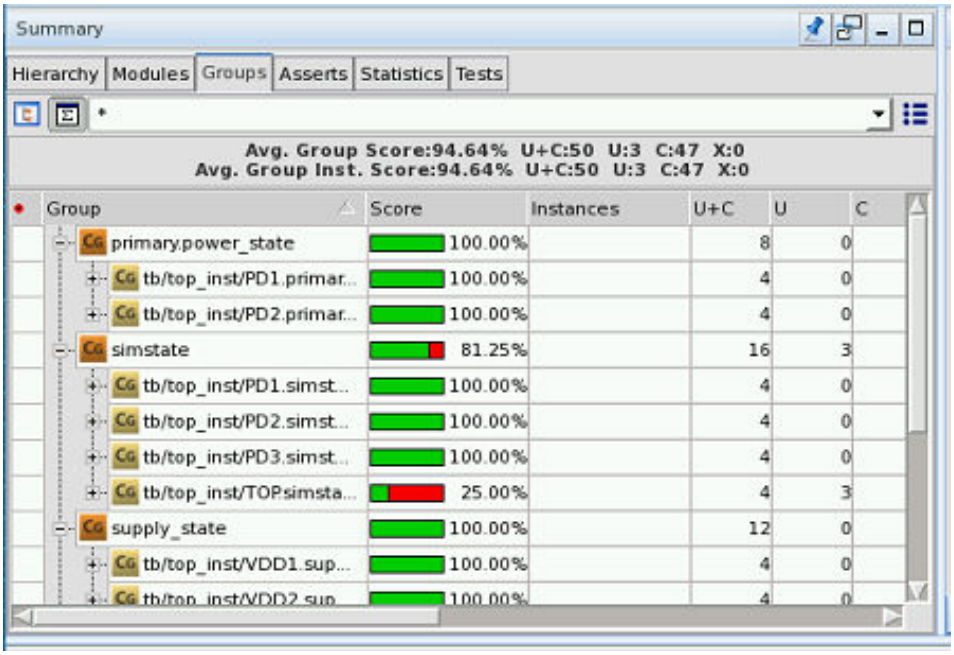
- PST states
- Power switch states/transitions
- Supply net/port states/transitions (root supply): States defined through the `add_port_state` command of the connected port and states inferred from the `-supply_expr` option of the `add_power_state` command for the connected supply set.

Feedback

- Supply set power state/transition (primary, `isolation_supply_set`, and `ret_supply_set`)
- Supply set simstate states/transitions (primary, `isolation_supply_set`, `ret_supply_set`, `default_isolation`, and `default_retention`)
- Isolation enable signal
- Retention SAVE/RESTORE signals
- Power switch control and ACK ports

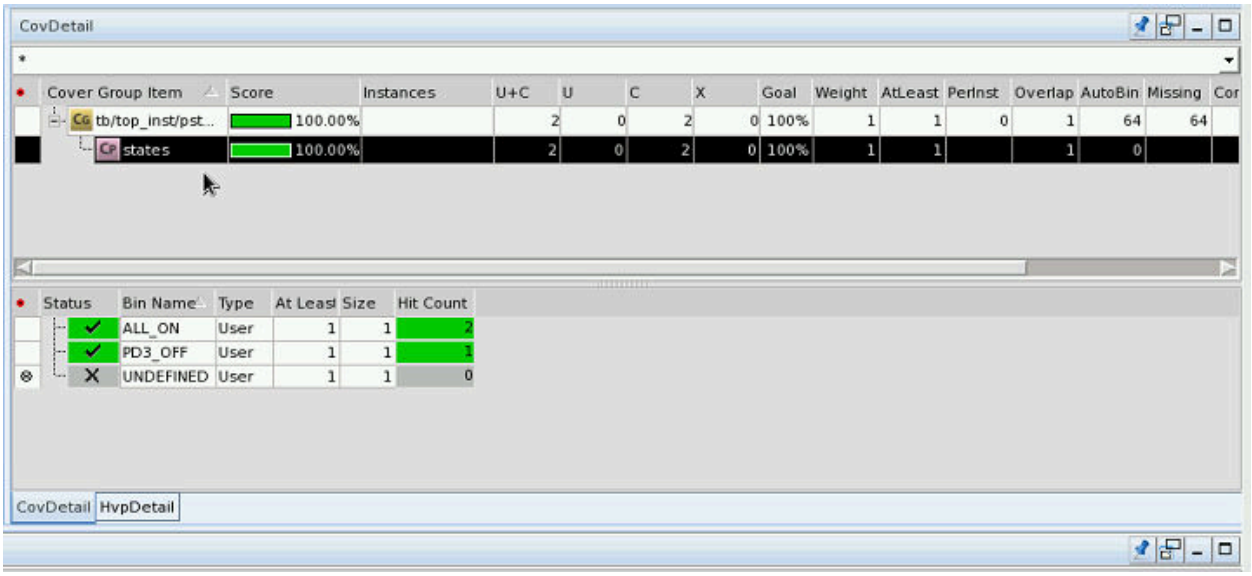
Figure 53 and Figure 54 shows covergroups and coverage details in the GUI.

Figure 53 Covergroups



Feedback

Figure 54 Coverage Detail



Enabling Coverage on Low-Power Objects

You can use the compile-time options listed in the following table to selectively enable coverage on the low power objects in a design. Then you can enable low power coverage from the `vcs` command line without the need to change the UPF.

Option	Description
`-power=cov_pd_simstate`	Enables power domain simstate coverage
`-power=cov_port_state`	Enables port state coverage
`-power=cov_implicit_supply_set`	Enables coverage on the implicit supply sets only (simstate and power state)
`-power=cov_explicit_supply_set`	Enables power state coverage for the explicit supply sets
`-power=cov_supply_set`	Enables all types of supply set coverage
`-power=cov_psw`	Enables power switch coverage on the following low power objects:• Power switch state (on/off/partial_on)• ACK port• Control port
`-power=cov_pst_state`	Enables only state coverage of PST
`-power=cov_pst_transition`	Enables only transition coverage of PST
`-power=cov_pst`	Enables Power State Table (PST) coverage. This option enables both the state and transition coverage for PST

Feedback

Option	Description
-power=cov_ret	Enables coverage on the retention save and restore signals
-power=cov_iso	Enables coverage on the isolation enable signal

You can specify more than one option on the command line using a plus sign (+) as a delimiter. For example,

```
% vcs -power=cov_psw+cov_implicit_supply_set -upf design.upf -sverilog \
 file_name.v
```

The above command enables coverage on the power switch and implicit supply sets.

Power-Aware Gate-Level Netlist Simulation Flow

In VCS Native Low Power (NLP), gate-level netlist simulations are not the same as RTL simulations. In addition to the design (gates and models) and the UPF, VCS NLP also needs .db files for library cells and the corresponding behavioral model for the cell. This section describes non-PG netlist simulation flow with VCS NLP.

Using Library Files for Simulation

Use the following search path and link library command in the configuration file. You can specify this file as an argument to the -power_config compile-time option.

```
db_search_path = db_file_directory db_link_library = {list_of_db_files}
```

To run the testcase:

```
% vcs <design_files> -v behavioral_model_file -upf file.upf \
 -power_config lp_config
```

Macro Cell Matching Criteria

The following conditions must be satisfied for the macro cell definition in the .db file to be termed as matched to the RTL behavioral model:

- Name of the Verilog model and the macro cell must match
- All RTL ports must match the macro cell logic pins in name and width
- Any RTL port matching macro PG pin must not have width greater than 1

Cell information is captured in the mvsim_native_reports/library_mapping.rpt report file. Matched and non-matched cells are listed in the mvsim_native_reports/library_match.rpt report file.

Feedback

Power-Aware and Non-Power-Aware Models

The model for a cell may or may not include PG pins. A model which does not contain PG pins is referred to as a “non-power aware” model. If PG pins are present in a model, they must exactly match what is present in the .db representation; otherwise VCS NLP treats it as a non-power aware model.

In a power-aware model, the PG pins can be:

- input/output ports of the model
- reg/wire
- supply_net_type
- supply0/supply1 type

Simulation Behavior for Power-Aware Models

If a model is recognized as power-aware, it means that the model itself completely describes the power behavior. That is, VCS NLP does not corrupt anything inside the model; it only drives the appropriate supply values to the PG pins of the cell. The model takes care of corruption.

For example, in the following power-aware model, the output pin `Z` is corrupted to `x` when the PG pin VDD is turned off.

```
module BUFFD0HVT (I, Z, VDD, VSS); input I, VDD, VSS; output Z; assign Z
= (VDD) ? I : 1'bx; endmodule
```

To force VCS NLP to corrupt the ports for all cells even when the models are power-aware:

```
set_design_attributes -attribute \ SNPS_override_pbp_corruption TRUE
```

VC LP Static Low-Power Multivoltage Rule Checking

The VC LP static low-power multivoltage rule checking tool verifies that the design conforms to the multivoltage rules and power intent specification. The types of checking it performs are divided into the following categories:

- UPF consistency and power architecture – Checks the isolation and level-shifting policies and elements versus requirements implied by the power state table; identifies missing, incorrect, and redundant isolation and level-shifting cells.
- Micro architecture – Island-order checks on control paths and clock paths in the design with respect to inserted power-related elements, including the following types of signals: isolation enable, power-gating enable, power switch acknowledge, retention

save/restore, clock reset, and scan enable. “Island ordering” refers to the always-on relationships between power domains that share control signals.

- Implementation versus intent – Checks the implementation versus UPF-specified intent for isolation, level shifting, retention, always-on synthesis, floating nets/pins, switch network, and PG connectivity.

Design Flow Stages and Multivoltage Checking

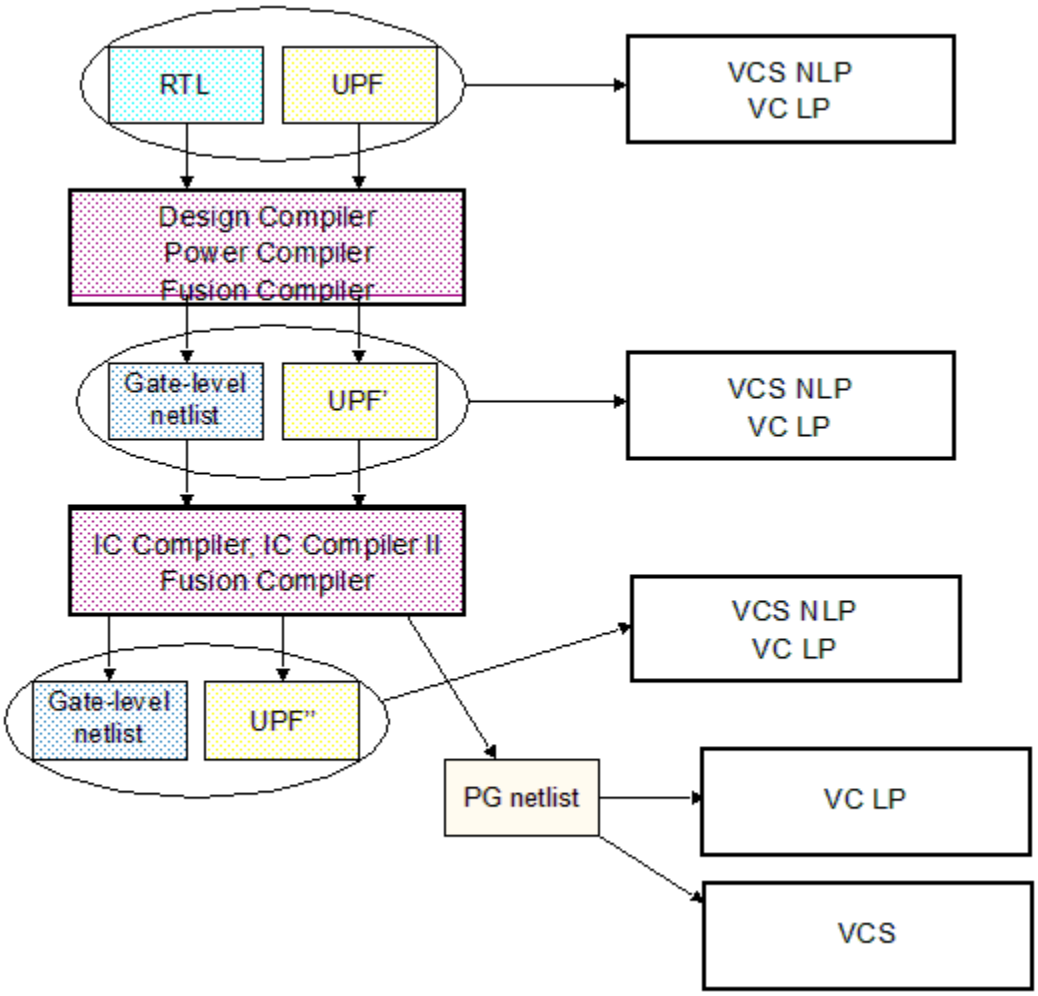
These are the stages in the design flow at which multivoltage simulation and rule checking can be performed:

- Original RTL + UPF, before synthesis
- Gate-level netlist + UPF’ produced by Design Compiler and Power Compiler tools
- Gate-level netlist + UPF” and PG netlist produced by IC Compiler, IC Compiler II, and Fusion Compiler tools

The types of checking that can be performed at each stage are summarized in Figure 55.

Feedback

Figure 55 Multivoltage Verification Flow



Original RTL + UPF

After you prepare an RTL description and UPF file for a design, and before synthesis, you can use VCS NLP and VC LP tools to verify the description for correct functionality and check for multivoltage rule violations. At this stage, the VC LP tool checks for the consistency of the UPF power intent with the power state table defined in the UPF. You perform this checking on the original RTL and UPF specification in the absence of any multivoltage cells (isolation cells, level shifters, retention registers, or power switches) or power and ground (PG) connections in the RTL. For complete architectural rule checking, you should specify the control signals such as clock and reset signals by using the `create_root` command.

This is a typical VC LP script used to check a design at the RTL + UPF level:

```
create_source -type clock -signal clk_n
create_source -type clock -signal rst_n
##All UPF checks are performed.
check_lp -stage upf
##Only isolation family checks performed.
check_lp -stage upf -family isolation
```

Gate-Level Netlist + UPF' from Design Compiler

After synthesis with the Power Compiler tool, and before placement and routing, you can perform simulation and verification of the gate-level netlist and modified UPF file, UPF' (UPF prime) generated during synthesis. The analysis operates on a netlist with isolation cells and retention registers, but without PG connections for logic cells and without power switches. For analysis at this stage, you provide the gate-level netlist, the UPF' file, the .db library models, the simulation models for standard cells, and multivoltage models for the isolation, level shifter, and retention cells.

The VCS NLP simulation performs the same type of analysis as at the RTL+UPF stage, but using the gate-level netlist instead of RTL and including the retention, isolation, and level-shifter cells inserted during synthesis. It verifies the multivoltage behavior of the synthesized logic and inserted power-related cells. You can optionally perform simulations using SDF annotation to account for different signal propagation delays under various power-down modes.

Similarly, the VC LP tool checks for conformity to the power intent specification and multivoltage rules using the generated gate-level netlist with inserted power-related cells. It checks the implementation against the UPF-specified power intent for isolation, level-shifting, and retention functions.

This is a typical VC LP script used to check a design at the gate-level netlist + UPF' after synthesis by the Power Compiler tool:

```
##By default, all design checks are performed.
check_lp -stage design
##Architectural checks are performed if roots are inferred from the
 design.
check_lp -stage design -family signalcorruption
```

Gate-Level Netlist + UPF'' and PG Netlist from IC Compiler (II)

After you complete placement and routing in the IC Compiler, IC Compiler II, or Fusion Compiler physical implementation tool, you can perform simulation and verification of the gate-level netlist and modified UPF file, UPF'' (UPF double-prime), generated for physical implementation. For analysis at this stage, provide the gate-level netlist, the UPF'' file, the .db library models, and multivoltage models for all the cells.

The PG-connected netlist has the power supply pins defined and connected as cell inputs and outputs, along with the logic pins, so it can be simulated with multivoltage simulation models by VCS directly. The simulation performs the same type of analysis as at the earlier stages, including the power switches inserted by the physical implementation tool. It verifies the multivoltage behavior of the synthesized logic and inserted power-related cells.

The VC LP tool checks for conformity to the power intent specification and multivoltage rules using the generated gate-level netlist with inserted power switches. It checks the implementation against the UPF-specified power intent for isolation, level-shifting, and retention functions. It also checks for always-on floating nets and pins, switch network functionality, and PG connectivity.

This is a typical VC LP script used to check a design after placement and routing by the physical implementation tool, using both the gate-level netlist + UPF'' and the PG-connected netlist:

```
##All PG connectivity checks are performed.
check_lp -stage pg
```

Logic Synthesis Using Design Compiler

The Design Compiler tool completely supports multivoltage logic synthesis, including automatic insertion of level shifters, isolation cells, and retention registers based on UPF commands and UPF-specified power state tables. A Power Compiler license is required for using UPF. The topographical mode of the Design Compiler tool accurately predicts post-layout timing and area in synthesis, helping to eliminate design iterations between synthesis and layout. DFT Compiler supports the use of retention registers and the insertion of level shifters and isolation cells where scan chains cross power domain boundaries.

For synthesis, the Design Compiler tool uses the RTL description of the design logic and the UPF description of the design power supply. The input UPF to the Design Compiler tool is used for RTL simulation and formal equivalency checking, as well as for synthesis. After it completes synthesis, the Design Compiler tool writes out a new gate-level UPF description, which includes the original UPF read in, any additional UPF commands run in the Design Compiler session, and UPF commands that describe the explicit supply connections to the multivoltage power management cells added during synthesis (isolation cells, level shifters, and retention registers). The UPF description generated by the Design Compiler tool can be used by compatible tools.

To synthesize a design with UPF power intent, the top-down flow is the most straightforward. These are the general steps in the flow:

1. Read the RTL file.
2. Use the `load_upf` command to read the UPF file.

Feedback

3. Specify the timing and power constraints.
4. Compile the design using the `compile_ultra` command.
5. Insert the scan chains using the `insert_dft` command.
6. Use the `save_upf` command to save the updated constraints into a new UPF file, which can be used as input to downstream tools.
7. Write the synthesis netlist into a file, which can be used as input to downstream tools.

When you use the `save_upf` command, the Design Compiler tool writes out UPF commands in the same order that they were read in with the `load_upf` command, and it writes the user-entered and tool-inserted UPF commands into a separate section. This feature makes it easier to track UPF changes by maintaining the command order of the original UPF script. However, this feature applies only to top-down synthesis; in a hierarchical synthesis flow, the `save_upf` command changes the ordering of the UPF commands.

Top-down synthesis is the simplest approach because you can compile the entire multivoltage design by running the compile command at the top level. For a large design, if necessary, synthesis can also be done using a bottom-up approach.

Power Domains and Hierarchy

Power domains defined for the design determine the connections of power nets and the implementation of power-down voltage areas. You can assign one or more hierarchical blocks to each power domain. Nested power domains are allowed as long as each power domain is contiguous and completely enclosed within a single higher-level power domain. Each hierarchical cell can belong to only one power domain.

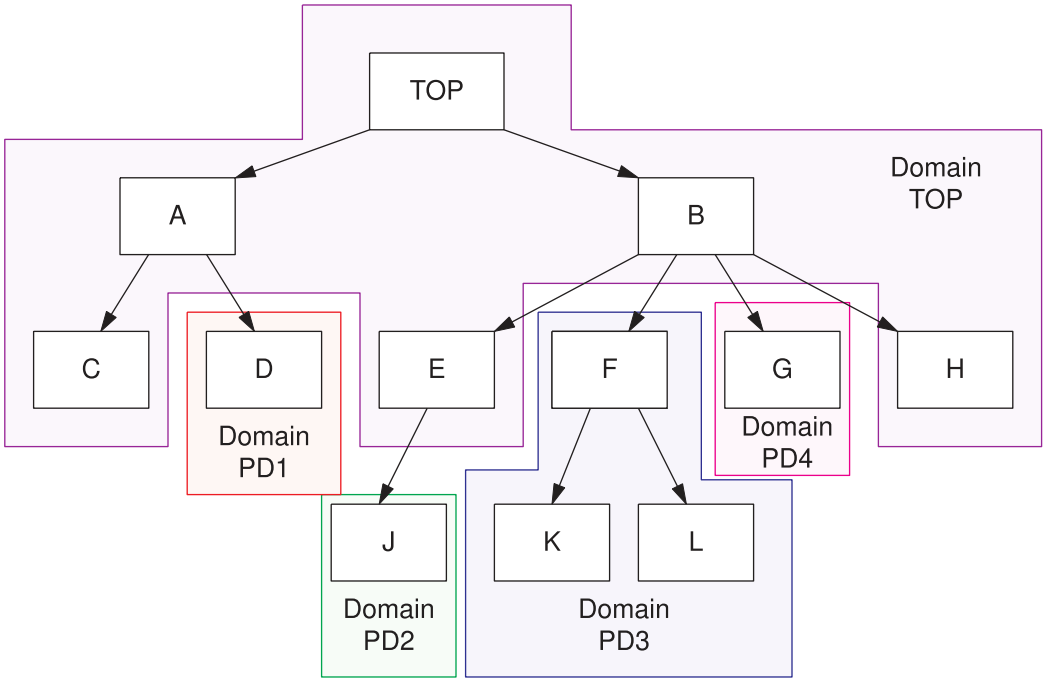
The following commands define the power domains of a design and assign hierarchical blocks to the domains:

```
create_power_domain TOP
create_power_domain PD1 -elements {A/D}
create_power_domain PD2 -elements {B/E/J}
create_power_domain PD3 -elements {B/F}
create_power_domain PD4 -elements {B/G}
```

The hierarchy and power domain membership of this example are illustrated in Figure 56.

Feedback

Figure 56 Power Domains and Hierarchy Example



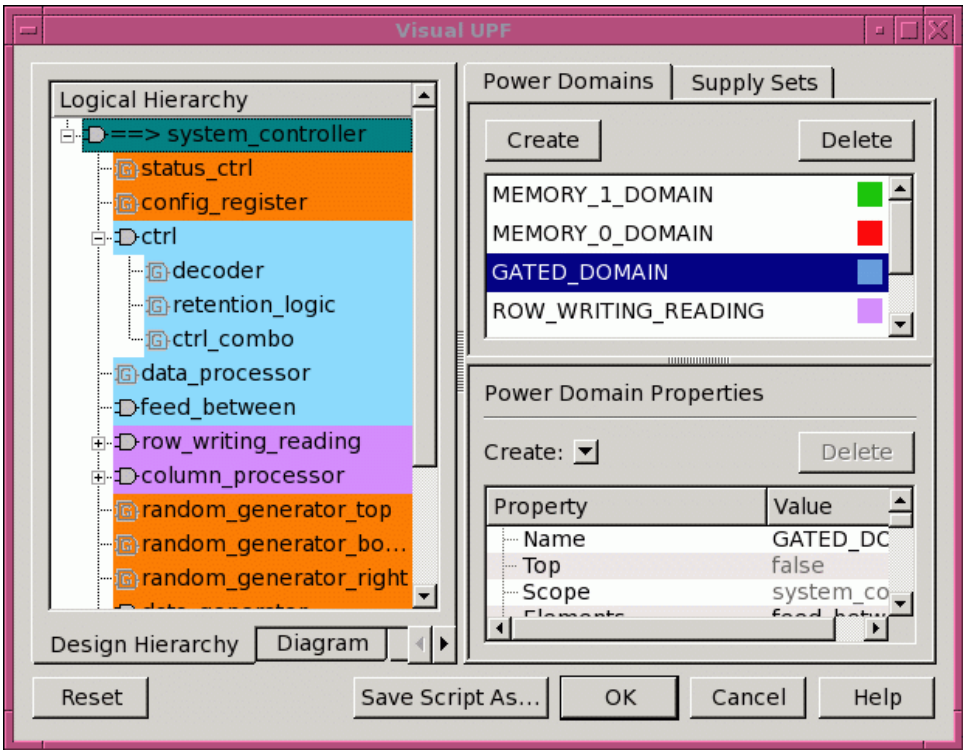
In this example, the first `create_power_domain` command assigns the TOP design to the power domain called TOP, which causes all the lower-level blocks to also belong to the same power domain. The subsequent `create_power_domain` commands assign some of the lower-level blocks to different power domains. Lower-level blocks not assigned to other power domains remain in the TOP power domain.

To find out about existing power domains that have been defined in the design, you can use the `report_power_domain` command, which generates a detailed report on the power domain characteristics, or the `get_power_domains` command, which returns a collection of power domains.

Another way to get power domain information is to use the Visual UPF dialog box in the Design Vision GUI. In the Design Compiler tool, use the `gui_start` command to open Design Vision. In the Design Vision window, choose Power > Visual UPF. This opens the Visual UPF dialog box showing a Logical Hierarchy view, color-coded by power domain, as shown in Figure 57.

Feedback

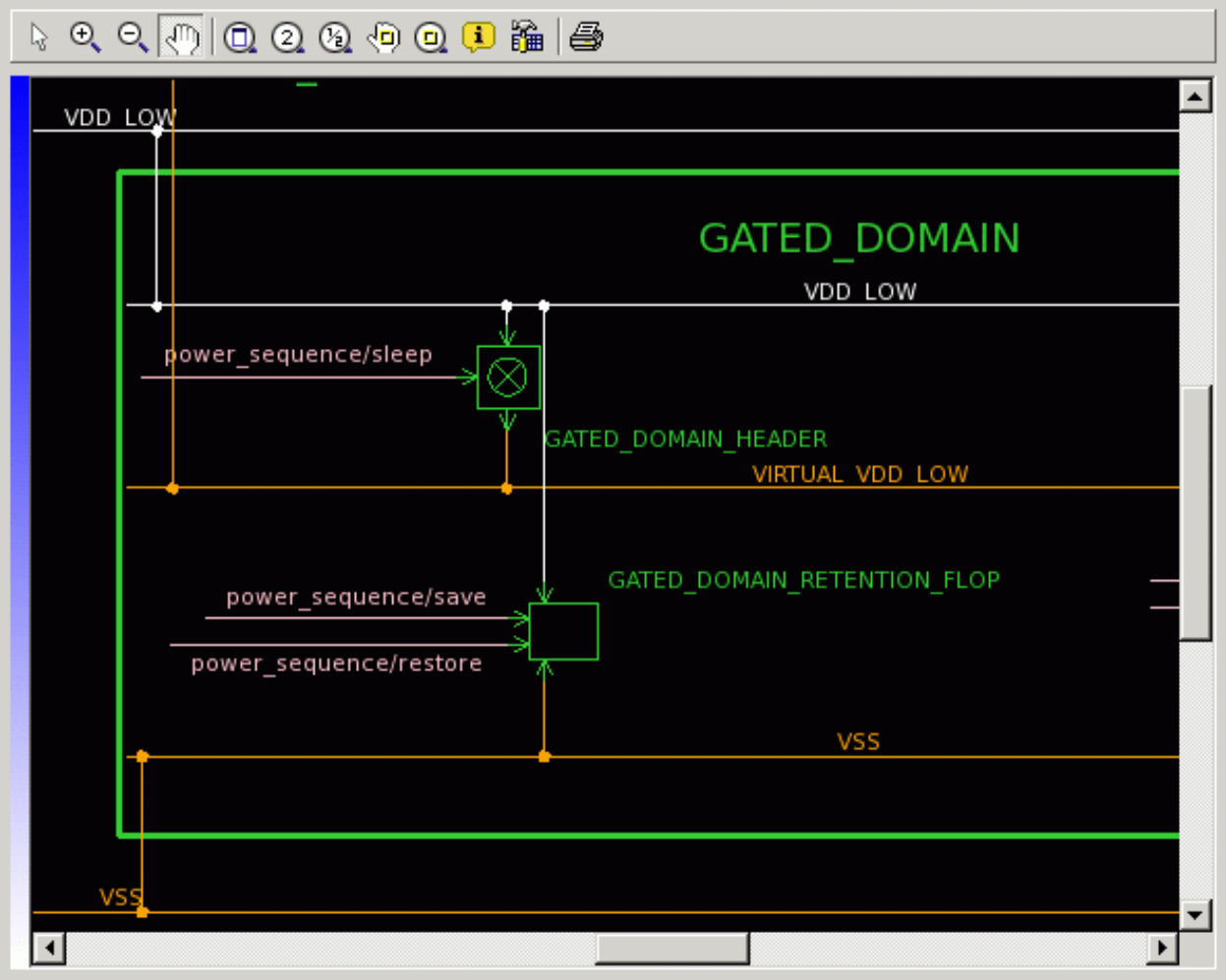
Figure 57 Visual UPF Design Hierarchy View



In the Visual UPF dialog box, click the Diagram tab to view the power network in a schematic diagram form as shown in Figure 58. You can pan and zoom the view in the same manner as a logic schematic.

Feedback

Figure 58 Visual UPF Diagram View

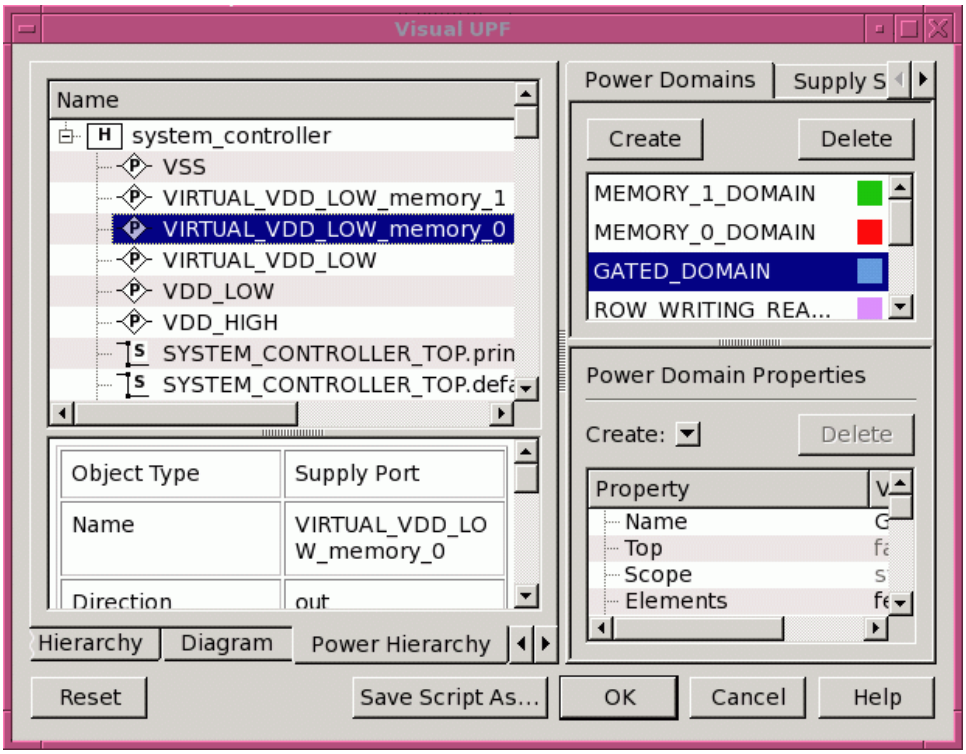


The diagram shows power domains, supply ports, power switches, isolation strategies, level-shifter strategies, and retention registers. However, it only shows the power intent specified by the UPF commands, not the specific implementation.

In the Visual UPF dialog box, click the Power Hierarchy tab to view the hierarchy of the power network in list format as shown in Figure 58. For each power domain, you can view a list of supply ports, supply sets, supply nets, isolation strategies, logic elements, and so on. You can select an item in the list to get detailed information about that item.

Feedback

Figure 59 Visual UPF Power Hierarchy View



You can also click the UPF Script command to view the UPF commands that define the current power network or the Error/Warning tab to view the current UPF warning messages.

Using the tabs and boxes on the right side of the Visual UPF dialog box, you can create, delete, and view information about the power domains and supply sets in the design.

For more information about using the GUI tools, see the online help in Design Vision (Help > Online Help, under Using UPF Multivoltage Tools).

Specifying Operating Voltages

You can specify operating voltages of supply nets in the design with the `set_voltage` command. Parts of the design powered by these supply nets are timed and optimized based on the cell properties at the specified voltages. Process and temperature values are determined by the top-level operating condition.

For example, the following commands specify the target libraries, operating conditions, and supply net voltages for the design:

```
## Target Libraries
set target_library "HVT.db LVT.db SVT.db"
```

Feedback

```
## Design operating condition name
set_operating_conditions WC09

## Set voltages on PN1 and PN2 supply nets
set_voltage 1.1 -object_list PN1
set_voltage 0.7 -object_list PN2
```

The voltages set on the supply nets must be consistent with the allowed voltages defined in the UPF power state table.

The `set_voltage` command allows minimum and maximum operating condition voltages to be specified for a supply net. If only one voltage is specified, it applies to the maximum operating condition. To specify voltages for both the maximum and minimum operating conditions, use the `-min` option, as in the following example:

```
set_voltage 0.86 -min 1.06 -object_list VDD
```

After execution of this command, the Design Compiler tool uses a library cell characterized at 0.86 volts for max (worst corner) analysis and at 1.06 for min (best corner) analysis.

Isolation, Level Shifter, and Retention Register Insertion

The Design Compiler tool automatically inserts isolation cells, level shifters, and retention registers during a compile operation. It uses the strategies set with UPF commands `set_isolation`, `set_isolation_control`, `set_level_shifter`, and so on, together with the power state table created by the commands `create_pst` and `add_pst_state`. The voltages set with the `set_voltage` command must be consistent with the voltages specified in the power state table.

Level shifters are the cells that function as the interface between power domains operating at different voltage levels. These cells ensure that the output transition of a driver can cause the receiving cell to switch even though the receiver is supplied with another voltage level.

Several options of the `set_level_shifter` command affect how buffer-type level shifters are added to the design. For example, the `-threshold` option establishes a threshold beyond which a voltage difference causes a level shifter to be inserted. The `-location` option specifies where the level shifters are placed in the logic hierarchy: `self`, `parent`, `fanout` (sinks), or `automatic` (the default; the tool chooses the best location for level shifter insertion).

After level shifters are inserted into the design, a `dont_touch` attribute is automatically set on the port-to-level-shifter net segment when the level shifter is within the block or on the level-shifter-to-port net segment when the level shifter is in the parent hierarchy.

Feedback

Level shifters should also be added to the clock tree nets. The clock tree synthesizer in the physical implementation tool assumes that all required level shifters have been added beforehand.

If level shifters are missing on particular nets, the timing engine maintains accurate timing analysis by automatically scaling transition times and delays according to the voltage on either side of the voltage interface.

The Design Compiler tool inserts isolation cells where signals leave a power-down domain. After an isolation cell is inserted, a `dont_touch` attribute is automatically set on the net segment connecting the isolation cell power domain interface. This protection prevents optimization from placing cells on that piece of net.

Enable level shifters are used when both isolation and level shifting are required. This typically occurs when the power domain can be shut down and has a different voltage. The approach for inserting enable level shifters is similar to that of isolation cells.

Always-On Synthesis

The cells of a shutdown block that are on a control signal path must remain active during the powered-down phase. These signals include power-down/power-up control signals and retention register save signals.

Always-on synthesis is performed by inserting buffers and inverters along the control signal paths using either of the following types of cells:

- Standard, single-power cells that are placed in special always-on site rows within the shutdown block's voltage area
- Special-purpose, dual-power cells that are marked as always-on

Use the `set_always_on_strategy` command to specify either the single-power or dual-power strategy for a given power domain. This is the command syntax:

```
set_always_on_strategy
  -object_list list_of_domains
  -cell_type single_power | dual_power
```

The dual-power strategy uses special-purpose, always-on cells. Each of these cells has two power pins: a primary pin that is hooked up to the primary supply and a backup pin that is hooked up to a supply net that ensures the cell will remain powered when the rest of the domain is shut off. The cell also has an attribute called `always_on` which is set to true at the cell level. This type of cell is known as a dual-rail, always-on cell.

The single-power strategy uses standard, single-power cells that are placed in dedicated always-on site rows of the power domain's corresponding voltage area. These always-on sites are powered by the backup power supply. During optimization, the logic synthesis tool does not make any changes to these always-on site rows, so it leaves these cells on

Feedback

the always-on paths. During placement in physical implementation tool, these cells are constrained to be placed in designated always-on site rows.

The dual-power strategy is preferred because always-on site rows are not needed and the cells can be placed anywhere in the voltage area.

During optimization, to improve QoR and reduce congestion, the tool uses regular buffers instead of always-on buffers on feedthrough nets when the primary power supply can be used to power the buffers, without introducing electrical violations. To give preference to load and driver supplies, do the following:

```
set_app_var mv_make_primary_supply_available_for_always_on false
```

Compile

Compile performs logic synthesis for the design, taking into consideration the multiple voltages and multiple power domains defined by UPF commands. Compile makes sure that the logic inside each power domain is mapped to technology cells from the correct voltage corner.

Design Compiler topographical mode completely supports all multivoltage features. The `create_voltage_area` command creates a voltage area for placing the cells associated with a particular power domain, like the same command in the IC Compiler, IC Compiler II, or Fusion Compiler tool. This command completes the physical constraints linked to the multivoltage design.

Note that the `compile_ultra` command, the default synthesis command in Design Compiler topographical mode, automatically ungroups all design hierarchies to achieve better timing results. However, the logic hierarchies associated with power domains are not ungrouped and the power domain boundary information is therefore preserved.

Apart from the regular optimization, compile also takes care of specific optimization and synthesis operations relating to level shifter, isolation, and retention cells.

Compile automatically inserts buffer-type level shifters on the appropriate nets and purges all redundant level shifters. By default, it does not insert level shifters on clock nets. To change this behavior, set the `auto_insert_level_shifters_on_clocks` variable to a list of clock names or to `all` before running compile.

Retention register inference and control port hookup operations are performed by compile. Compile maps user-specified registers in the shutdown power domain with retention registers. Compile synthesizes applicable registers with retention registers and hooks up the save and restore pins to control ports. Library retention cells should have specific attributes at cell and pin level to identify the retention cell and the “save” and “restore” control pins.

If you want to insert power management cells outside of a compile operation, you can do so with the `insert_mv_cells` command. This command inserts isolation, level-shifter, and enable level-shifter cells based on the UPF definition, without doing a full compile.

Multivoltage Checking

It is important to check the design signal consistency across power domains. There are some features that can change the hierarchical ports of the logic netlist, and if those ports are defined at the power domain interface, then new isolation cells or level shifters might be needed. The following commands can be used to perform such checks:

- `check_mv_design`
- `analyze_mv_design`

You can also examine and debug multivoltage designs in the Design Vision GUI using the MV Advisor violation browser. When you select a violation, the violation browser displays a detailed report that includes an explanation of the warning or error message, debugging information, and suggestions for fixing the violation. For details, see Design Vision Help and the *Power Compiler User Guide*.

The Design Compiler tool also has other reporting capabilities that are relevant for multivoltage designs. The most useful commands are `report_timing -voltage`, which returns the voltage value for each cell in the selected timing path, and `report_hierarchy`, which provides a summary of the voltage applied to the hierarchical blocks in the design.

Here are some examples:

```
report_timing -attributes -transition_time -capacitance \
              -voltage -nosplit
report_hierarchy -nosplit -noleaf
```

DFT Methodology Using DFT Compiler

If a scan chain crosses power domain boundaries, additional level shifters and isolation cells are required on the scan path to take care of voltage shifting and isolation requirements. This affects design timing and area quality of results. Therefore, the most common requirement for multivoltage DFT methodology is to make sure scan insertion is bounded by the power domains. When it is necessary for two or more power domains share the same scan chain, the number of domain crossing points should be minimized.

When you run DFT Compiler in the Design Compiler tool, any necessary multivoltage special cells (level shifters and isolation cells) are automatically inserted according to the strategies specified by UPF commands. The insertion strategy specifies the types of level shifters or isolation cells that are needed when a scan path crosses a power domain boundary and the locations where these cells are inserted.

Feedback

Scan Insertion Configuration

By default, DFT Compiler clusters the scan chains within a power domain. This clustering reduces the number of power domain crossings and therefore the number of level shifters and isolation cells needed in the scan path. However, to allow the tool more flexibility in creating scan chains, you can allow power domain crossings by using the following commands:

```
set_scan_configuration -power_domain_mixing true
set_scan_configuration -voltage_mixing true
```

When power domain mixing is enabled, a scan chain can span multiple power domains. Similarly, the voltage-mixing option allows a scan chain to mix cells from different voltage levels. Even when power domain mixing and voltage mixing are enabled, DFT Compiler still attempts to keep the scan chains within each power domain as much as possible, thus minimizing the number of crossings to other power domains.

For the `insert_dft` command to properly insert level shifters and isolation cells, you must specify the level shifter and isolation cell strategies on a power-domain basis, even if you have specified similar strategies on blocks and individual ports. By specifying level shifting and isolation strategies at the power domain level, the tool knows what strategy to use for any new ports that it needs to create between power domains.

Another way to specify the scan chains is to group the registers. DFT Compiler offers grouping features that allow specific selection of all the registers to be inserted in the chains. This is not an automatic feature, but it gives you full control of scan insertion in all power domains. The `set_scan_group` command can be used to group all the registers in a particular power domain that should be clustered:

```
set_scan_group scan_group_name -include_elements [list sequential_cells]
```

You can use the `set_scan_path` command to restrict the cells to a particular scan chain, You can specify either a list of cells or scan groups, or you can identify cells by clock name using the `set_scan_path` command. For example:

```
## Sequential cells reg1, reg4 and reg5 will be grouped
## together in single scan chain, possibly with other registers
set_scan_group SG1 -include_elements [list reg1 reg4 reg5]

## If you would like to further restrict and have only these 3 sequential
## cells in a single scan chain, you can specify the following
set_scan_path -include_elements [list SG1] -complete true
```

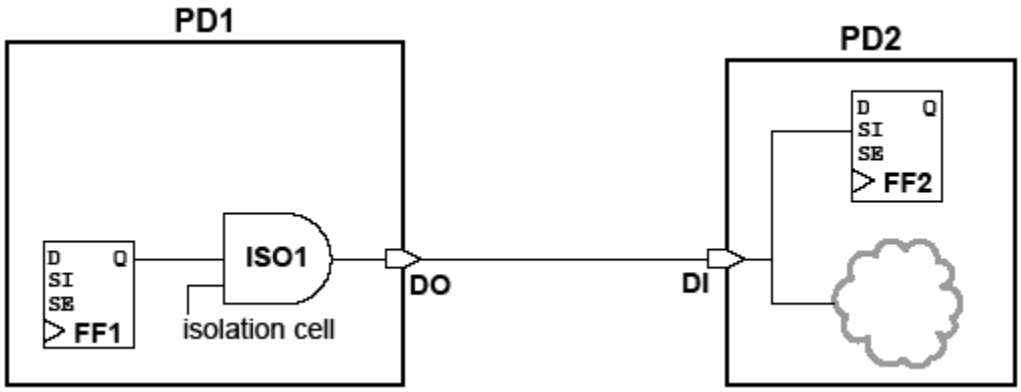
If running the `set_scan_path` command results in violating the voltage mixing or power domain mixing specification, the `preview_dft` and `insert_dft` commands issue warning messages about the violations.

Feedback

Scan Insertion

The `insert_dft` command works with power domains, level shifters, and isolation cells. Scan insertion does not remove any existing level shifters and isolation cells. If the multivoltage management cell is within a block, the `insert_dft` command reuses the existing hierarchical port. This behavior is illustrated in Figure 60. The same isolation cell serves to isolate both the normal logic signal and the scan data signal leaving the output of FF1.

Figure 60 Scan Insertion With Reused Isolation Cell

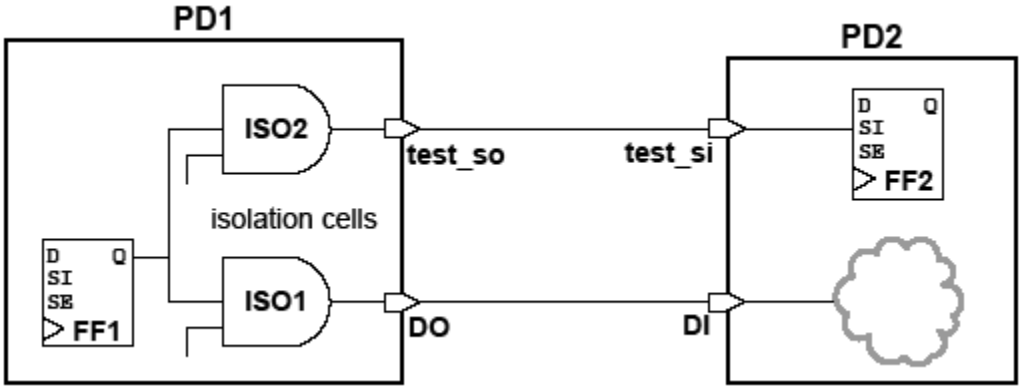


To prevent the `insert_dft` command from reusing multivoltage management cells, use the following command:

```
set_scan_configuration -reuse_mv_cells false
```

In that case, DFT Compiler creates two isolation cells and two ports, as shown in Figure 61.

Figure 61 Scan Insertion Without Reusing Isolation Cells



Feedback

During the insertion of the scan chains, it is very common for new hierarchical ports to be created in the design to connect the scan chains across the logic hierarchy. These new hierarchical ports are protected by the insertion of level shifter and isolation cells as needed.

The multivoltage functionality for adaptive scan in DFT Compiler is the same as that of standard scan. The compressor and decompressor logic is created in the correct voltage corner at the top level. Also, the scan insertion (for both internal scan and scan compression test modes) respects the power domains and special cells. Typically, adaptive scan creates numerous internal scan chains, each crossing the power domain boundary, which can increase the design area.

The `check_mv_design` command reports missing level shifters and isolation cells at the power domain interface. For a detailed report, use the `-isolation`, `-level_shifters`, and `-verbose` options in the command.

Retention Registers

If retention registers are used in the design, the states of the retention save and restore signals must be constrained to the powered-up state for DFT implementation. The following commands set these constraints:

```
set_dft_signal -view existing_dft -type Constant \
 -port SAVE -active_state non-save-value
set_dft_signal -view existing_dft -type Constant \
 -port RESTORE -active_state non-restore-value
```

For information about DFT and multivoltage designs, see the “Multivoltage Support” section of the *DFT Compiler User Guide*.

Design Planning Using IC Compiler

You can use the IC Compiler physical implementation tool for design planning of flat physical designs. This includes basic floorplanning capabilities such as design initialization, power planning, and fast placement of standard cells and macros.

Design planning for multivoltage designs involves some additional considerations. The major floorplanning tasks in multivoltage designs are creating voltage areas, routing multiple supply nets, inserting multiple-threshold CMOS (MTCMOS) power switch cells, and analyzing the supply network.

Power Domains and Voltage Areas

It is important to include UPF-specified power domain and supply net objects in the physical implementation phase. When implementing multivoltage designs, these objects

are required to create voltage areas, to derive interface net constraints, and to perform specific optimizations.

Power domains and voltage areas should maintain a one-to-one mapping relationship. A power domain and its matching voltage area should have the same name and same set of hierarchical cells. When a power domain is mapped with the voltage area, the associated logic information is automatically derived from power domain.

Physically nested voltage areas are supported as long as the inner voltage area is completely contained within the outer voltage area. A nested voltage area corresponds to a logically nested power domain. Voltage areas that would overlap physically must be resolved into nonoverlapping, abutted rectangular or rectilinear subareas.

The floorplan is the first design stage where the voltage area can be defined. If you cannot identify the best location for a voltage area, you can run a quick placement and trace the voltage-area-related logic to determine a suitable location. On the other hand, if you know an ideal location for a voltage area, you can specify its coordinates explicitly.

Feedback

To create a voltage area, use the `create_voltage_area` command or the equivalent Floorplan > Create Voltage Area dialog box. In a UPF flow, when you use the `-power_domain` option of the `create_voltage_area` command, the cells belonging to the specified power domain are automatically assigned to the voltage area. in that case, you do not use the `-name` option and you do not list of modules assigned to the voltage area, as that information comes from the power domain definition.

For example:

```
create_voltage_area -power_domain MULT \
   -coordinate {50.140 50.140 300.00 210.00} \
   -guard_band_x 2 -guard_band_y 2
```

Both rectangular and rectilinear shapes are supported for voltage areas. For an explanation of these command options, see the *IC Compiler Implementation User Guide*.

The specified voltage area is the physical placement area for the logic belonging to the corresponding power domain. Except for level shifters, all cells associated with a voltage area operate at the same process, voltage, and temperature (PVT) operating condition values.

If you define voltage areas during floorplanning, the floorplan must contain a default voltage area for the top-level cells and at least one other voltage area. If you directly create the voltage areas by using the `create_voltage_area` command, the tool automatically derives a default voltage area for the top-level leaf cells and level shifters that do not belong to any block.

Supply nets, which are not part of a logic netlist, are required to create power routing. Multivoltage designs typically deal with multiple power and ground nets. This means power routing is always a challenging task for multivoltage designs. The supply net objects, created along with power domains, help in this task by guiding the creation and connection of real supply nets.

Supply net objects can also help identify the power and ground connections to be made between the standard cell power pins inside the power domain and the real supply net. Using this information, the `place_opt`, `clock_opt`, and `route_opt` commands make incremental power connections automatically for any new cells added during optimization. You can also explicitly make these connections by using the `derive_pg_connection` command.

The IC Compiler tool can automatically create power straps and rings for the voltage areas. Starting from reliability constraints, power network synthesis automatically generates power supply routing for certain design regions, such as voltage areas. In addition, power network analysis provides voltage drop and electromigration information.

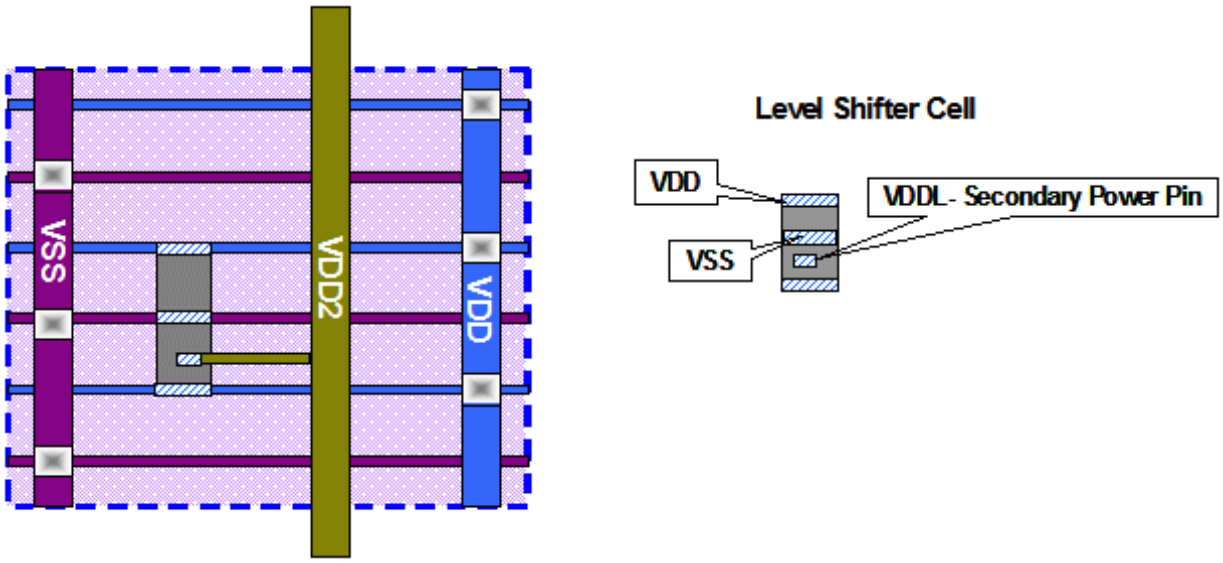
Feedback

Power Management Cell Placement

Power-related cells such as level shifters, isolation cells, retention registers, always-on cells, and power switches typically have multiple power and ground pins and are taller than standard cells. As a designer, you need to make sure that all the power pins are connected and correctly routed to the power nets.

For example, consider the low-to-high level-shifter cell with two power pins shown in Figure 62. The cell is taller than a standard cell and has 2X height.

Figure 62 Level Shifter With Dual Power Pins



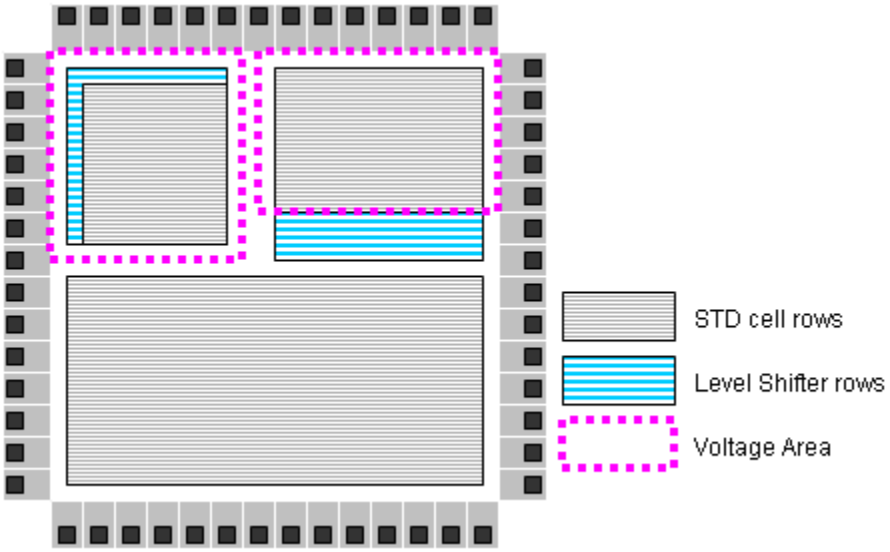
The secondary power pin, VDDL, should be connected to the power supply net VDD2. For power supply routing, VDDL needs to be routed to the VDD2 strap. However, this cannot be achieved with the normal signal or power routing method. A special type of routing, called net mode routing (specified with the command `preroute_standard_cells -mode net`), is required to connect the secondary pin to the nearby power strap.

The layout of the special cell in Figure 62 is a simple and effective form. The dual height and power-pin layout of the cell allows the special cell to be freely placed along with the standard cells in core area. The timing-driven, congestion-driven placement automatically takes care of placing the cell properly.

On the other hand, level shifter or isolation cells might have more complex structures. A possible way to accommodate these special cells in the floorplan is to place them by themselves in a dedicated area. However, degraded quality of results can be expected due to the imposed placement restriction. Figure 63 shows an example of what separate level-shifter placement areas might look like.

Feedback

Figure 63 Multivoltage Floorplan



Power switch cells require special placement near the power straps. The `create_power_switch_array` command is used to insert and place the switch cells automatically on the specified insertion grid. After placing the switch cells, the sleep net, which connects all the sleep pins of switch cells, is connected by using the `connect_power_switch` command.

Another aspect of design planning is the selection of the always-on cell strategy. If always-on cells with dual power pins are used for always-on synthesis, these cells can be placed along with standard cells. However, if normal buffers (with a single power supply) are used in always-on paths, you need to make sure that the cells in the always-on path are pulled into a specific placement area which has constant power supply. This type of special placement area for always-on cells is called an exclusive move bound.

A move bound can be defined as exclusive bound and assigned with the always-on cells, as shown in the following example:

```
## SINGLE POWER RAIL AO STRATEGY
set_always_on_strategy -object_list {MODULE_A} -cell_type single_power

## CREATE MOVE BOUND FOR AO CELLS
create_bounds -name AO_BOUND \
              -exclusive \
              -coordinate {120 140 160 180}

set_power_guide -name AO_BOUND
```

For more information about always-on marking and implementation, see the *IC Compiler Implementation User Guide*.

Feedback

Power Planning

After you complete the design planning process and have a complete floorplan, you can perform power planning with the following commands:

- `create_rectangular_rings` (Preroute > Create Rings in the GUI) adds power and ground rings to the power network.
- `create_power_straps` (Preroute > Create Power Straps) adds power and ground straps connected to the rings.
- `preroute_standard_cells` (Preroute > Preroute Standard Cells) connects power and ground pins of standard cells to the straps and rings of the power mesh.
- `synthesize_fp_rail` (Preroute > Synthesize Power Network) connects power and ground pins of standard cells to the straps and rings of the power mesh.

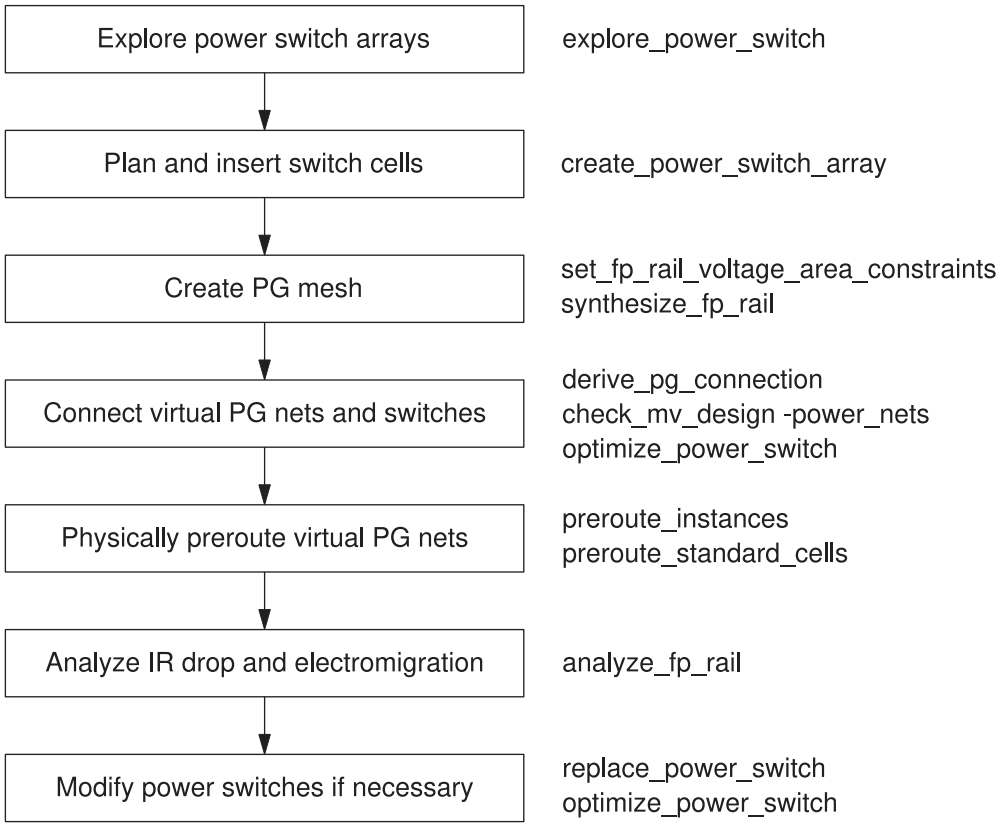
A design with power gating requires a network of power-switching cells distributed physically around or within each power-down block. This network consists of PMOS header switches between VDD and the block power supply pins or NMOS footer switches between VSS and the block ground pins. The following commands support the planning of power switch placement:

- `map_power_switch` (a UPF command) specifies which power switch cell is to be used to implement a specified switch instance in a specified power domain.
- `explore_power_switch` can be used to explore and estimate the placement of power-switching cells in conjunction with a virtual power and ground network.
- `create_power_switch_array` defines a 2-dimensional grid on which the power-switching cells are placed in the floorplan and adds the cells to the logical netlist.
- `optimize_power_switch` resolves IR drop problems and optimizes the power-switching cells by automatically sizing the cells larger where required to meet IR drop requirements and sizing them smaller where IR drop requirements are exceeded.
- `connect_power_switch` connects the power-switching control signals to the switch inputs of the power-switching cells.

Figure 64 shows the typical flow for planning power switches.

Feedback

Figure 64 Power Switch Planning Flow



The `analyze_fp_rail` command performs power network (IR drop) analysis for both real and virtual power nets to the power-switching cells, for complete or incomplete power nets. The `synthesize_fp_rail` performs power network (IR drop) analysis for both real and virtual power nets to the power-switching cells, for complete or incomplete power nets.

The `set_fp_rail_strategy` command sets various parameters for power network synthesis and power network analysis such as macro connection and blocking, strap alignment, chimney checking, power pad checking, and completeness checking. The `set_fp_rail_constraints` command sets the constraints for power network synthesis, including power strap layer constraints, power ring constraints, and global constraints. The `set_fp_rail_voltage_area_constraints` command sets the power network synthesis constraints for a specified voltage area.

The `derive_pg_connection` command performs automatic power/ground connection for power/ground pins of cells of the design. The `-reconnect` option causes any existing power/ground connections to be replaced with new ones.

Feedback

The `commit_fp_rail` commits the power network, including power and ground wires and vias, based on the results of power network synthesis.

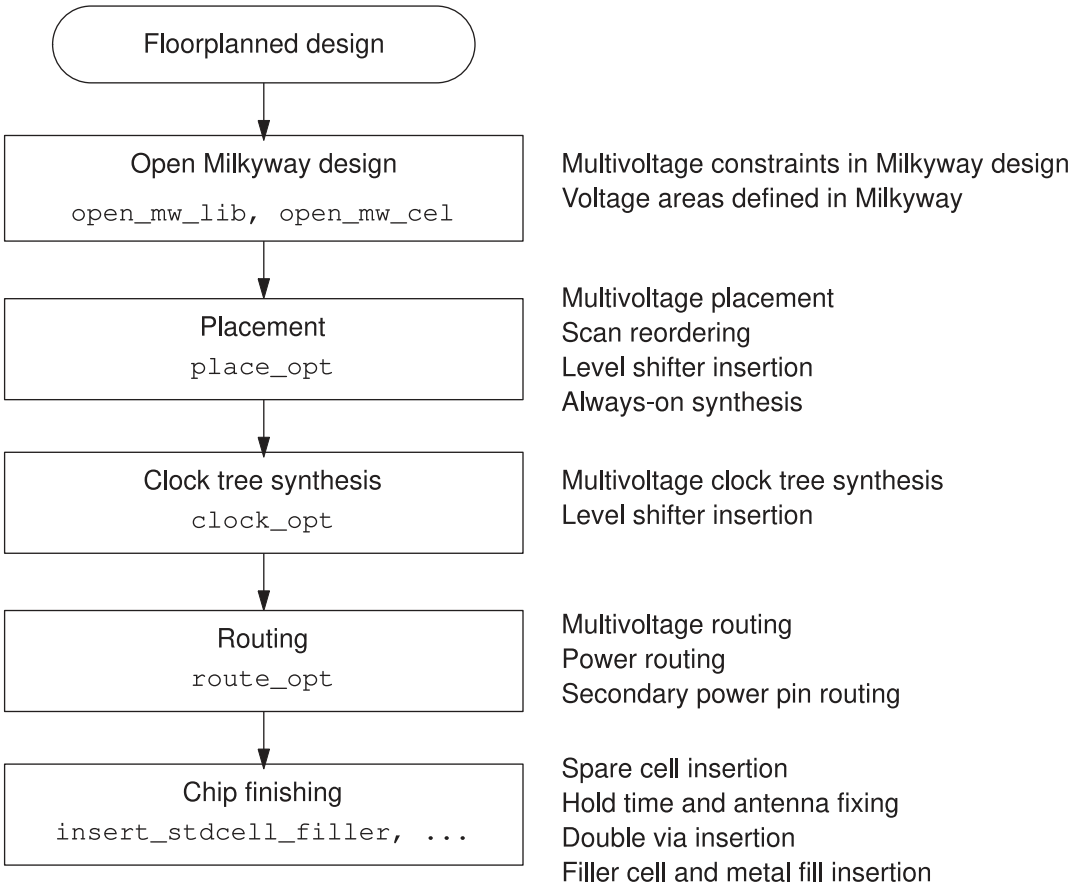
For more information about power network analysis and power network synthesis in design planning, see the *IC Compiler Design Planning User Guide*.

Physical Implementation Using IC Compiler

After the floorplan is completed with the rows, power planning, and I/O placement determined, physical synthesis and implementation can begin. The IC Compiler tool provides a broad set of features to perform a physically flat implementation of the design.

Figure 65 shows the general steps in the physical implementation flow and the low-power support features of the flow.

Figure 65 IC Compiler Low-Power Physical Implementation Flow



Feedback

Starting with a design previously floorplanned in IC Compiler, read in the design from the Milkyway database. The design already has the multivoltage constraints and voltage areas previously defined in the Design Compiler and IC Compiler tools. Then perform multivoltage placement, clock tree synthesis, and routing. Finally, perform the chip finishing steps.

In IC Compiler, use `set_operating_conditions` for the top level and `set_voltage` for all power supplies of the design:

```
icc_shell> set_operating_conditions opcond_name

icc_shell> set_voltage value -object_list { supply_nets }
```

Before performing optimization, perform a multivoltage design check with the `check_mv_design -verbose` command. This command checks the multivoltage constraints, isolation requirements, and connection rules. It generates detailed reports on any violations.

The core commands `place_opt`, `clock_opt`, and `route_opt` can optimize timing by adding new instances to the correct logic hierarchy, and can legally place standard cells, level shifters, and isolation cells while honoring the voltage areas.

IC Compiler automatically preserves any existing level shifters, isolation cells, and enable level shifters that are already part of the design. If required, level shifters cells can also be incrementally inserted with the `insert_level_shifters` command. Enable level shifters and isolation cells can be inserted with the `insert_isolation_cell` command. However, the introduction of level shifters or isolation cells should occur as early as possible in the design flow to help maintain control of critical timing paths and avoid timing problems at the implementation stage.

The ability to recognize the cells at the power domain boundary is the key to appropriately setting the `dont_touch` attributes. IC Compiler recognizes the level shifters, isolation cells, and enable level shifters automatically and propagates the `dont_touch` attributes to the nets that must preserve the voltage consistency of the design.

IC Compiler places level shifters, isolation cells, and enable level shifters close to the voltage area boundaries to reduce the routing lengths crossing the voltage areas. it sets `dont_touch` attributes on those wires to preserve them.

The power domain and voltage area definitions can be reported at any time with the `report_power_domain` and `report_voltage_area` commands. If new power domains or voltage areas need to be defined in IC Compiler, they can be created with the `create_power_domain` and `create_voltage_area` commands.

The following commands are often used for checking and reporting multivoltage designs. For detailed information about any command, see its man page.

```
check_design
check_isolation_cells
```

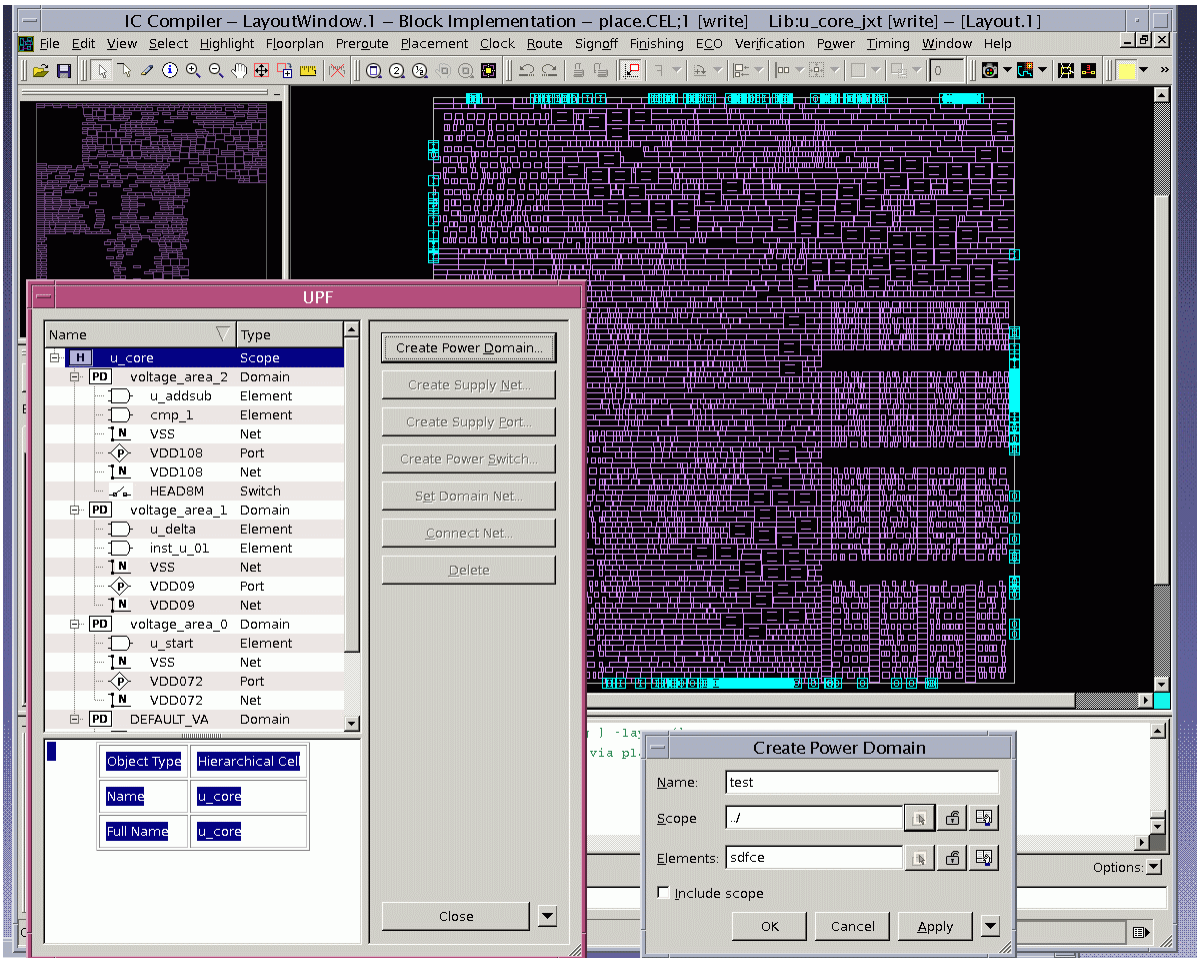
Feedback

```
check_mv_design
report_cell
report_delay_calculation
report_hierarchy
report_isolation_cell
report_level_shifter
report_operating_conditions
report_power_domain
report_power_switch
report_retention_cell
report_supply_net
report_supply_port
report_target_library_subset
report_timing
report_voltage_area
```

The graphical user interface (GUI) provides a means to view and edit the specified power supply structure. Use Power > UPF Browser to open the UPF dialog box. In the dialog box, you can view the current UPF configuration and make changes such as creating power domains and connecting supply ports. See Figure 66.

Feedback

Figure 66 UPF Dialog Box in IC Compiler



In black-box hierarchical flows, when some supply nets are used in subblocks but not at the top level, you can use the variables `power_nets_for_upf_supply` and `ground_nets_for_upf_supply` to identify named nets as power supply nets or ground nets. By identifying the functions of these nets, the tool has the information it needs to derive the new supply nets at the top level.

Placement and Optimization

When the `place_opt` command performs placement and physical synthesis, it takes into account the voltage areas, power domains, instance-based targeting, always-on logic, and special cells protecting the power domain interfaces (isolation cells, level shifters, and enable level shifters). This means that each design instance is assigned to a single voltage area, the voltage area is exclusively allocated to a certain logic hierarchy partition,

Feedback

and any new instance created for a certain logic hierarchy is legally placed within the voltage area boundary and subject to the same operating conditions.

In addition, automatic high-fanout net synthesis (part of the `place_opt` command) honors voltage areas. Buffer trees are built with dedicated subtrees for the fanout cones in each voltage area. In particular, automatic high fanout synthesis selects buffers according to the associated operating condition. After running the `place_opt` command, it is good practice to check whether new level shifters are needed, because new hierarchical ports might have been created.

The IC Compiler tool minimizes the formation of nets that cross multiple voltage area boundaries. When buffering a net that crosses a voltage area boundary, the tool divides the net into segments, with each segment confined to a single voltage area, and restricts buffer insertion to these segments. It also ensures that a tie cell in one voltage area does not feed a cell in another voltage area.

The tool respects the logic hierarchy when handling the tie-high and tie-low cells. It inserts tie cells by default as long as a `dont_touch` or `dont_use` attribute is not in the library.

Placement recognizes the cells working at the power domain interface and places those cells near the related voltage area boundary. Moreover, the tool's routing estimation is capable of detouring around voltage areas. Therefore, the `place_opt` command can correctly optimize the design logic within a voltage area as well as the logic outside the area.

As noted earlier, level shifters can be taller than the standard cells. If the height of the level shifter is double or triple the height of a standard cell, then it is recommended to have only one type of row where all standard cells and level shifters can be placed together. However, the possibility of placing level shifters side by side with standard cells also depends on the layout of the level shifter. Due to DRC rules, in some cases level shifters might require special placement attention.

All the core commands in the IC Compiler tool perform always-on synthesis for the selected always-on paths. The methodology is the same as in the Design Compiler tool. In addition, if the single-power-rail strategy is used with an always-on exclusive bound, the optimizations make sure that the newly added always-on cells are automatically pulled into the always-on bound.

When multicorner-multimode technology is used, optimization works on the worst timing violations across all scenarios, thus eliminating the convergence problems observed in sequential approaches. Optimization can be performed for DRC, setup and hold, or setup only.

Scan Chain Reordering

Another step performed in the physical synthesis stage is scan chain reordering, which can reduce the scan wire lengths and congestion and improve routability. To carry out scan

reordering, the physical implementation tool needs to read in the SCANDEF file generated by the Design Compiler tool, which contains the scan chain definitions. After that, the `place_opt` and `clock_opt` commands can reorder the scan chains based on the physical locations of the registers. For example,

```
## READ SCANDEF
read_def -verbose ./dft.scan.def
..
## RUN PLACE_OPT WITH SCAN REORDER
place_opt -optimize_dft
...
## RUN CLOCK_OPT WITH SCAN REORDER
clock_opt -optimize_dft
```

The SCANDEF file defines groups of scan chains called partitions. Each partition contains one or more scan chains that are allowed to exchange flip-flops. A partition can have only one operating condition and one power domain.

You can optionally use the `set_optimize_dft_options` command to set the parameters for DFT optimization before using the `place_opt` command. To perform scan chain optimization as a separate process, use the `optimize_dft` command.

Level Shifters

Level shifters should be created in the netlist as early as possible, in the synthesis stage using the Design Compiler tool. The `compile_ultra` or `compile` command automatically inserts level shifters where required.

In some cases, the IC Compiler tool must add level shifters on new paths that cross voltage area boundaries, for example, those resulting from scan chain insertion, test port hookup, or logical feedthrough insertion. You can insert level shifters with the `insert_level_shifters` command. Before you do so, be sure that the voltages have been set on the design with the `set_voltage` command, the level shifter libraries have been specified in the `target_library` variable, and the level shifter strategy and threshold have been defined.

A typical level insertion flow consists of commands similar to the following:

```
set_level_shifter_threshold -voltage 0.1
set_level_shifter_strategy -rule all
check_mv_design -level_shifters
insert_level_shifters
```

The `check_mv_design -level_shifters` command checks and reports level shifter violations involving nets, cells, and operating conditions. You can run this command at any point in the flow.

To remove level shifters that have been inserted by the IC Compiler tool, use the `remove_level_shifters` command.

Feedback

Isolation

An isolation cell selectively drives a known signal value at the edge of a shutdown voltage area. An enable-type level shifter performs both level shifting and isolation functions. During logic synthesis, the Design Compiler tool can replace isolation cells with enable level shifters if the appropriate cells are available in the libraries.

Isolation cells can be instantiated at the RTL level, or they can be manually inserted by using the `insert_isolation_cell` command throughout the flow. Enable level shifters cannot be inserted by a user command, but they can be inserted at the RTL level or inserted automatically during compilation to replace existing isolation cells.

There are several ways that isolation cells and enable-type level shifters might be connected to the input and output nets of the voltage areas. It is possible to unintentionally introduce redundant isolation cells into the RTL description or to fail to specify necessary isolation cells. You can use the `check_isolation_cells` command to check for these conditions.

Always-On Synthesis

An always-on path is a control net that is driven from an always-on power domain and ends within a shutdown power domain, and which must remain on during shutdown. Some examples of always-on signals include the control pins of power switch cells, isolation cell enable signals, and feedthrough nets.

IC Compiler automatically recognizes the always-on characteristic of enable pins of special-purpose, dual-power cells and the input pins of single-power standard cells placed in always-on power wells. This includes isolation cells, enable level shifters, power switch cells, and retention registers. Other types of always-on signals might need to be manually marked. IC Compiler determines the always-on paths by tracing back from the always-on pins along the fanin logic cone of each pin.

During execution of the `place_opt` command, IC Compiler performs optimization of always-on paths, including buffering and cell sizing. It uses ordinary cells for the portion of an always-on path outside of the shutdown power domain, and it uses always-on cells for the portion within the shutdown power domain.

During optimization, to improve QoR and reduce congestion, the tool uses regular buffers instead of always-on buffers on feedthrough nets when the primary power supply can be used to power the buffers, without introducing electrical violations. To give preference to load and driver supplies, do the following:

```
set_app_var mv_make_primary_supply_available_for_always_on false
```

IC Compiler performs always-on synthesis only when the target library contains an always-on buffer and an always-on inverter. The target library can have special-purpose cells

marked for use in always-on paths. Alternatively, you can identify library cells or pins in your IC Compiler session by setting the library cell or library pin attributes. For example, you can use the following syntax:

```
set_attribute [get_lib_pins lib_name/cell_name/pin_name] always_on true
```

IC Compiler can use either single-power or dual-power cells for always-on synthesis. Placement of standard, single-power cells is restricted to the site rows of special-purpose, always-on power wells created within the shutdown block's voltage area. Dual-power cells can be placed anywhere in the shutdown power domain, but the selected locations must allow for the routing of secondary power lines.

To use single-power cells, you need to set the always-on strategy to single-power mode, create an exclusive move bound to contain the new cells, and define the move bound as an always-on power guide. Here is a typical script to perform these steps:

```
set_always_on_strategy -object_list PD1 \
  -cell_type single_power
create_bounds -name AON_RFF_BUFS \
  -coordinate {547 131 573 893} \
  -exclusive [list [get_always_on_logic]]
set_power_guide -name AON_RFF_BUFS
```

To get information about power guides that have been defined, use the `report_power_guide` command. To remove a move bound as a power guide, use the `unset_power_guide` command.

The `check_mv_design -connection_rules` command reports always-on problems such as a normal cell driving an always-on net in a power-down domain, an always-on cell driving a normal net, and certain types of pass gate connection problems.

If an always-on net passes through a shutdown voltage area and needs to be buffered, always-on buffers must be used in the shutdown area. To allow IC Compiler to generate these types of feedthroughs, use a flow similar to the following command sequence:

```
set_fp_voltage_area_constraints -allow_feedthroughs true
...
#  Global routing here
...
create_voltage_area_feedthroughs -preview_feedthroughs
set_fp_voltage_area_constraints -allow_feedthroughs true \
  -exclude_feedthroughs [get_nets nets_to_exclude]
create_voltage_area_feedthroughs -finalize_feedthroughs
set_fp_voltage_area_constraints -allow_feedthroughs false
...
place_opt
...
clock_opt
...
route_opt
```

Feedback

Clock Tree Synthesis

The `clock_opt` command considers voltage areas. No special setup is required. The clock tree synthesis engine in IC Compiler recognizes the logic hierarchy associated with the voltage area and creates the clock tree bottom-up by clustering sink points from the same voltage area. After the clock subtrees are built for each voltage area, clock tree synthesis joins the subtrees at the root of the clock net.

Here is an example of a clock tree synthesis script:

```
## SET CTS OPTIONS
set_clock_tree_options -max_transition 0.5 -max_capacitance 0.6

## CLOCK TREE REFERENCES
set_clock_tree_references -references {BUF2 BUF4}

remove_clock_uncertainty [all_clocks]

## CLOCK_OPT
clock_opt -fix_hold_all_clocks
```

For bottom-up clock tree synthesis, IC Compiler performs endpoint clustering based on voltage areas. It builds a separate clock subtree for each voltage area, without crossovers between these subtrees.

You should insert level shifters and isolation cells as needed on clock nets that cross power domains before running clock tree synthesis. To check for proper insertion, use the `check_mv_design -level_shifters` and `check_mv_design -isolation` commands.

The `clock_opt` command supports multicorner-multimode technology. Clock tree synthesis is performed only in the clock scenario and optimizations are performed for all scenarios.

Routing

By default, IC Compiler observes the following rules while routing multivoltage designs:

- Nets connecting within a single voltage area are restricted to that voltage area.
- Nets connecting cells outside a voltage area must detour around that voltage area.
- Net that cross voltage area boundaries are minimized.

Feedthroughs that cross voltage areas are allowed if you execute the following command:

```
set_fp_voltage_area_constraints -allow_feedthroughs true
```

To perform power and ground routing to standard cells, use the `preroute_standard_cells` command. This command connects power and ground pins

Feedback

in the standard cells to the power and ground rings or straps, and it connects power and ground rails in the standard cells.

The `preroute_standard_cells` command has a `-mode` option, which lets you select one of three power/ground connection modes: `rail`, `tie`, or `net`:

- Use `preroute_standard_cells -mode rail` to connect standard cells by rails.
- Use `preroute_standard_cells -mode tie` to connect standard cells by tying all the power and ground pins to nearby targets.
- Use `preroute_standard_cells -mode net` to connect the pins of standard cells to a nearby power ring or strap; additional command options let you specify the layers, widths, and maximum allowed fanout for the connections. Each routing cluster has a limited number of secondary power pins connected to the nearest power strap.

The net mode is useful for routing the secondary power pins of dual-power cells. You can use the `-nets` option to restrict the routing to specified power and ground nets.

Multicorner-Multimode Technology

Multicorner-multimode technology is a key capability required to optimize “dynamic voltage and frequency scaling” types of multivoltage designs. Using multicorner-multimode technology, IC compiler can optimize and analyze the design concurrently for all its operating voltage and frequency states.

In IC Compiler, a design scenario refers to a functional mode, a corner, or a combination of both functional mode and corner. With concurrent optimization and analysis, all scenarios are handled simultaneously for timing, area, design rule checking, signal integrity, and power.

A multivoltage design that exhibits multiple voltage states can be treated as a multicorner design (one corner for each set of operating voltage states). Scenarios are created by using the `create_scenario` command. A scenario definition must be followed by setting the scenario-specific constraints: operating conditions, Synopsys Design Constraints (SDC), and TLUPlus models. For example,

```
## CREATE SCENARIO S0
create_scenario S0
source ./cstr_S0.sdc
set_operating_conditions -max WCCOM -min BCCOM
set_voltage 1.08 -object_list {VDD VDDM VDDMS VDDI VDDIS VDDG VDDGS}

## CREATE SCENARIO S0
create_scenario S1
source ./cstr_S1.sdc
set_operating_conditions  -max WCCOM -min BCCOM
set_voltage 1.08 -object_list {VDD VDDM VDDMS}
```

Feedback

```
set_voltage 0.865 -object_list {VDDI VDDIS VDDG VDDGS}
set_voltage 0 -object_list {VSS}
```

For multivoltage designs, it is important to protect the nets connecting level shifter and isolation cells to the power domain boundary in all scenarios. In this case, the tool preserves the `dont_touch` attribute on these nets across all scenarios. You can run the `check_mv_design` command on a per-scenario basis and then insert level shifters or isolation cells if required.

Timing and Signal Integrity

The IC Compiler timing engine considers voltage areas. It can detect those signals that connect the driver and load across power domains that operate at different voltage levels. Where level shifters are not inserted, the timing engine scales the transition time according to the voltage levels of the domains involved.

Delay calculation and transition scaling details can be obtained by using the `report_delay_calculation` command, as shown in the following report example:

```
Rise Delay Voltage Adjustment
   Driver voltage: 1.08
   Load voltage: 1.32
   Driver delay trippoint: 50
   Load delay trippoint: 50
   Driver transition time: 1.25402
   Driver upper trippoint: 90
   Driver lower trippoint: 10

Adjustment = driver_transition_time * (load_voltage *
load_delay_trippoint - driver_voltage * driver_delay_trippoint) /
(driver_voltage * ( upper - lower ))
    = 1.25402 *
(1.32 * 50 - 1.08 * 50) / (1.08 * ( 90 - 10 )) = 0.174169
Adding 0.174169 to original rise delay (0) gives 0.174169
```

The `report_timing` command has an option, `-voltage`, that shows the operating conditions used by the timing engine for each instance in the timing report.

During multicorner-multimode analysis, timing analysis is carried out on all scenarios concurrently, and costing is measured across all scenarios. As a result, the timing and constraint reports show worst-case timing across all scenarios.

The timing engines in IC Compiler and PrimeTime take into account the voltage impact during crosstalk analysis. The timer individually scales voltage levels based on the delay contributions of aggressors and victims. Signal integrity analysis and optimization are supported in both multicorner and multimode flows.

Feedback

Saving the Design in the Milkyway Database

When the physical implementation is complete, the design can be saved into a CEL view. By default, the `save_mw_cel` command saves all the scenarios into the CEL view.

The Milkyway CEL can be read into StarRC to generate parasitic information in SPEF format. The Verilog netlist and Synopsys Design Constraints (SDC) file should be written out for signoff static timing analysis in PrimeTime. For example:

```
change_names -rules verilog -hierarchy

write_script -nosplit -output ./DB/route.script
write -format verilog -hierarchy -output ./DB/route.v
write_link_library -nosplit -output ./DB/route.link

current_scenario S0
write_sdc -nosplit ./DB/route_S0.sdc

current_scenario S1
write_sdc -nosplit ./DB/route_S1.sdc

save_mw_cel mydesign
```

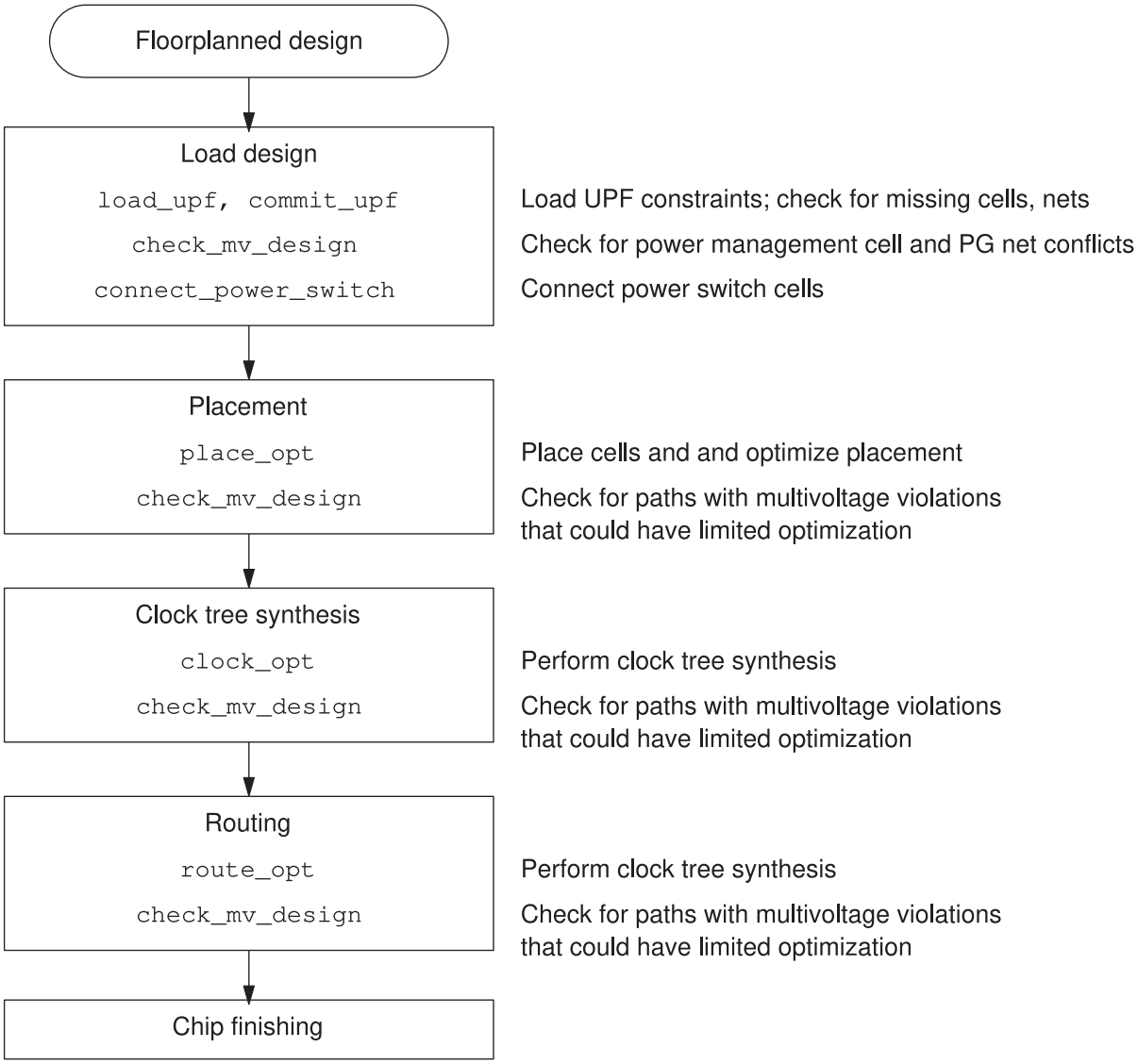
Physical Implementation Using IC Compiler II and Fusion Compiler

After the floorplan is complete with site rows, power planning, and I/O placement, you can proceed to physical synthesis and implementation. Fusion Compiler provides a broad set of features to perform a physically flat implementation of the design.

Figure 67 shows the general steps in the physical implementation flow and the low-power support features of the flow.

Feedback

Figure 67 IC Compiler II and Fusion Compiler Low-Power Physical Implementation Flow



Reference Library Setup

The IC Compiler II and Fusion Compiler tools need to have always-on buffers and inverters available in the reference libraries. If the usage of these library cells is restricted by the `set_lib_cell_purpose` command, the usage purpose must include

Feedback

`optimization` and `cts` so that the cells can be used for optimization and clock tree synthesis. For example,

```
prompt> set_lib_cell_purpose -include none \
  [get_lib_cells my_lib/invAO*]
...
prompt> set_lib_cell_purpose -include {optimization cts} \
  [get_lib_cells my_lib/invAO*]
...
```

The purpose settings are stored in the lib_cell attributes called valid_purposes, included_purposes, and excluded_purposes. Another lib_cell attribute, always_on, must be true for always-on library cells. You can query the attributes of a library cell as follows:

```
prompt> report_attributes -application -class lib_cell my_lib/invAOx1
...
Design            Object     Type       Attribute Name     Value
-------------------------------------------------------------------
my_lib/invAOx1    ...        boolean    always_on          true
...
```

In the IC Compiler II or Fusion Compiler tool, to query the attributes of cell instances in the design, you can take advantage of the cascaded attributes feature. You can use any collection-type attribute to query an attribute of that collection class. For example, all instances have a ref_block attribute, which can be used to access the lib_cell class attributes of the instance:

```
prompt> get_attribute [get_cells U882] ref_block.valid_purposes
optimization cts
```

For information on modeling power management cells in the Liberty description language, see Library Requirements and the *Library Compiler User Guide*.

Loading UPF Constraints

Loading UPF constraints for a design consists of two steps:

1. `load_upf` – Loads the UPF commands into memory
2. `commit_upf` – Applies the complete, finalized UPF commands to the current block

You can load multiple UPF files in memory before you commit them. The `commit_upf` command performs global checks for UPF consistency; resolves PG conflicts among the netlist, floorplan, and UPF specification; and associates power strategies with existing multivoltage cells. It also tries to resolve any conflicts between the power intent specified in the UPF file and the actual power and ground nets in the design.

Feedback

Only a very limited subset of UPF commands are allowed to be used after the `commit_upf` command. If you need to make significant changes to the UPF infrastructure of the design, you need to use the `reset_upf` command and start over with the `load_upf` command.

The UPF files can only contain standard Tcl and UPF commands. Synopsys Tcl collections (created by `get_cells`, `get_pins`, and so on) are not allowed in UPF files. However, you can assign Tcl collections to Tcl variables before you load UPF, and use the Tcl variables inside the UPF file as needed. For example,

```
prompt> set pd1_elements [get_cells * -filter is_hierarchical==true]
{u0 u1 u2 u3 um up}
prompt> load_upf myUPF.upf
...
prompt> commit_upf
...
```

The UPF file can contain commands that use the pd1_elements variable:

```
...
create_power_domain PD_1 -elements $pd1_elements
...
```

Checking the Power Intent

Checking the multivoltage-related infrastructure with the `check_mv_design` command is recommended after the commit-UPF, placement, clock tree synthesis, and routing steps of the flow. The `check_mv_design` command performs a complete check of UPF intent and PG connectivity for the block and generates a detailed report on any violations found.

The `check_mv_design` command checks the following power-related issues:

- The UPF has been committed.
- HighConn and LowConn connections on lower-domain boundaries are correct and consistent with the PG supplies.
- Each net has a driver, and each pin has an associated and connected supply.
- The cross-domain paths have the power management functions (isolation and level shifting) required by the power state table.
- Unnecessary redundant power management cells have been optimized out of the design.
- The enable signals of isolation cells and enable level shifters are connected and driven by cells with appropriate always-on functionality.
- Switch cells and tie-off pins are correctly connected.

Feedback

Existing Power Management Cells

The IC Compiler II and Fusion Compiler tools automatically preserve existing power management cells already existing in the design: level shifters, isolation cells, enable level shifters, power switches, and retention registers. The ability to correctly associate existing power management cells with the UPF strategy is a key part of preventing multivoltage violations during physical optimization.

The IC Compiler II and Fusion Compiler tools dynamically maintain the multivoltage logic to allow maximum flexibility during logic optimization and clock tree synthesis. The tool performs automatic association by tracing the logic that crosses power domain boundaries after usage of the `commit_upf`, `create_mv_cells`, and `associate_mv_cells` commands.

Automatic association fails if the defined strategy in UPF does not completely match the implementation. You can debug association failures by using the following command:

```
prompt> report_mv_path -cell cell_instance_name
```

After a power management cell has been associated by the `commit_upf` or `create_mv_cells` command, the cell has an attribute called preregistration which is set to the name of the associated strategy.

You can force an association by a cell and a power management strategy by using the `set_power_strategy_attribute` command. However, you should do this only if you understand the reason for the failure of automatic association and you want to override the default association.

Inserting New Power Management Cells

You can insert additional level shifters, enable level shifters, and isolation cells by using the `create_mv_cells` command in accordance to the power strategies defined in the UPF. To determine the power strategies for a power domain, use the `get_power_strategies` command.

You should introduce level shifters and isolation cells as early as possible in the design flow to ensure accurate timing of critical paths and to avoid unexpected timing problems later during the flow. You can perform optimization and clock tree synthesis only on nets that are free from multivoltage violations.

Level shifter strategies are required where signals cross from one domain to another at different supply voltages. The tool can insert new level shifters only when a valid strategy has been defined. If level shifter strategies are missing in the UPF, you can generate a generic level shifter strategy by using the `create_mv_cells` command.

```
prompt> create_mv_cells -generate_strategy level_shifter
...
```

Feedback

Before you insert level shifters based on the UPF strategies, be sure that the level shifter cells are available in the reference libraries and the voltages have been set for all supplies at each operating corner.

You can search for level shifter library cells with the `get_lib_cells` command:

```
prompt> get_lib_cells -filter is_level_shifter
```

To set the voltage for an operating corner, use the `set_voltage` command:

```
prompt> set_voltage -corners C1 0.900
```

To limit the choice of level shifter library cells used, define the level shifter mapping by using the `map_level_shifter_cell` command in the UPF file for each strategy. To apply voltage thresholds limits for level shifter insertion, use the `set_level_shifter` command with the `-threshold` option, as in the following example:

```
...
set_level_shifter LSup1 -domain PD1 -threshold 0.12 ...
map_level_shifter_cell LSup1 -domain PD1 -lib_cells {LSa1 LSa2}
...
```

After you define and apply the strategies and constraints, use the following command to insert the level shifters into the design:

```
prompt> create_mv_cells -level_shifter
...
```

When the tool inserts a level shifter, it sets the dont_touch attribute on the net between the port and the level shifter.

To insert an enable level shifter, ensure than an isolation cell is inserted. If there is a voltage difference between the two domains, the tool's optimization automatically converts the isolation cell into an enable level shifter.

The tool's core commands, `place_opt`, `clock_opt`, and `route_opt`, consider multivoltage constraints as they operate. They maintain the correct power domain scope by adding new instances as needed during timing optimization, DRC optimization, and clock tree synthesis. They also fully respect physical boundaries of voltage areas for all instances, including the placement and legalization of power management cells. In general, to minimize routing lengths crossing the voltage areas, the tool places power management cells close to the voltage area boundaries.

Specifying UPF Constraints for Physical-Only Cells

The UPF constraints for physical-only cells can cause issues when the UPF file is read into other tools that do not support physical-only cells, such as verification tools.

Feedback

Therefore, you can surround the UPF constraints for physical-only cells with the following syntax:

```
if {[info exists snps_handle_physical_only] && \
     $snps_handle_physical_only}
{
    connect_supply_net VDDS -ports [get_pins FILLER*/VDD]
}
```

By default, the `snps_handle_physical_only` variable is set to `true` in the IC Compiler II and Fusion Compiler tools. When you load and commit the UPF file, the constraints are applied to the physical-only cells.

When you save a UPF file with the `save_upf` command, the tool uses the same syntax for the user-specified and tool-derived UPF constraints for physical-only cells. Therefore, any tool that does not have the `snps_handle_physical_only` variable set to `true` ignores these constraints.

Reporting and Debugging Power Intent

Several commands are available in the IC Compiler II and Fusion Compiler tools to report and debug power intent. The following table provides a summary.

Table 3 Commands for Reporting and Debugging Power Intent

Command	Report content
`check_mv_design`	Main checking command. Performs all checking necessary to ensure power intent matches actual design.
`report_mv_path`	Path debugging command reports details on all multivoltage constraints. Good for finding association problems and multivoltage net violations
`check_bufferability`	Determines whether nets can be buffered in a voltage area. Debugs buffering issues.
`report_cells -power`	Reports PG pin connection of a cell.
`check_pg_connectivity`	Verifies physical PG network connectivity.
`report_power_domain`	Reports UPF intent related to power domains.
`report_pst`	Reports UPF power state table information.

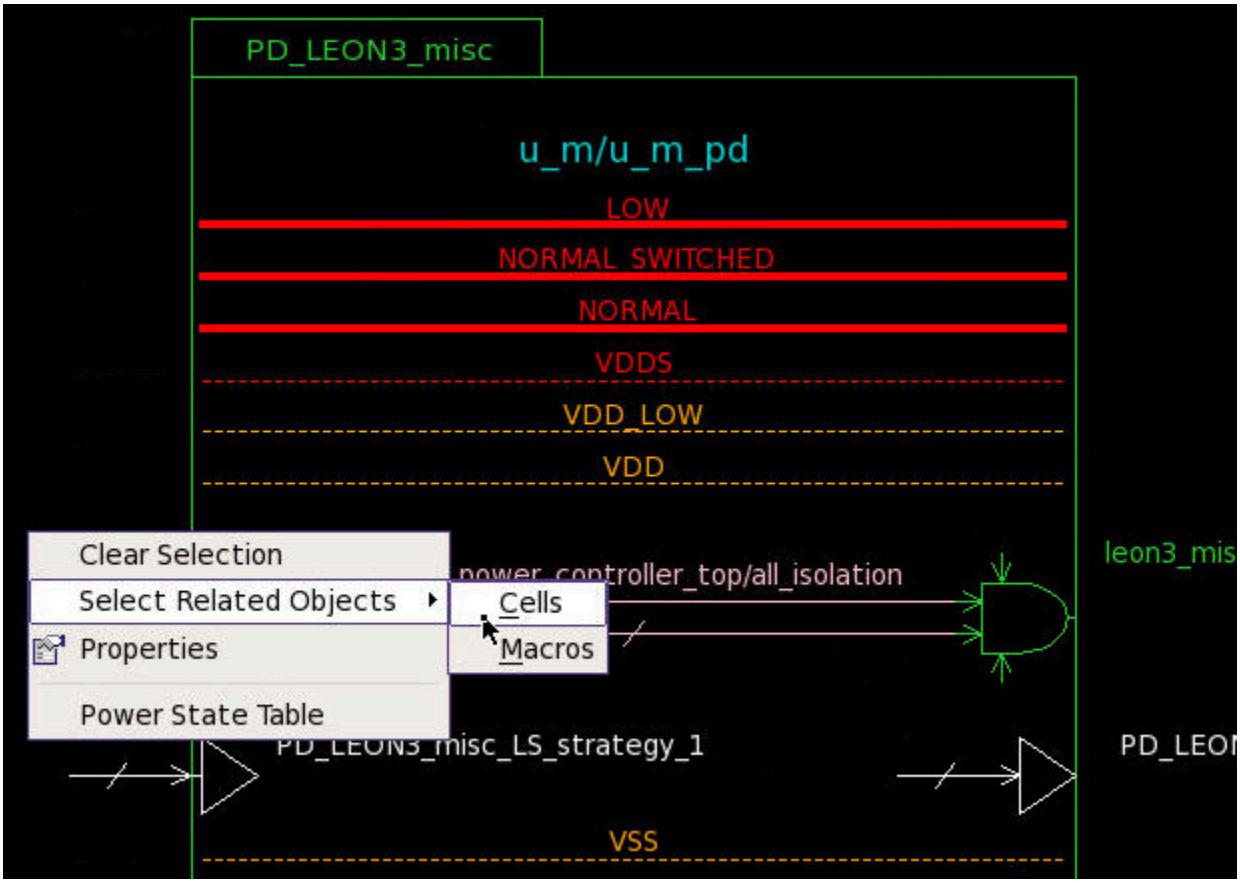
You can use get_* commands with filtering to get information about power-related objects in the design. For example, to get a collection of all always-on physical cells in the design:

```
prompt> get_cells -physical_context -filter ref_block.always_on==true
```

Feedback

You can also use the graphical user interface. From the GUI menu bar, choose View > UPF Diagram View to open the UPF diagram. This diagram helps you quickly understand the power intent of the design. It supports cross-highlighting of various objects in other views. For example, you can right-click the object of interest and choose the Select Related Objects option, as shown in the Figure 68.

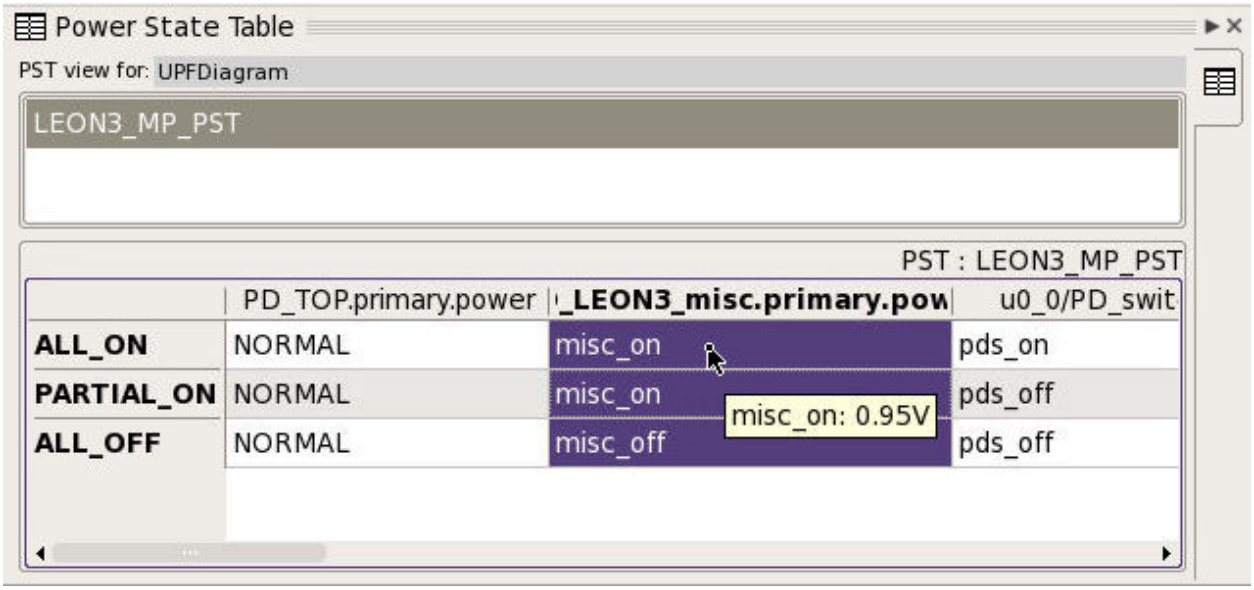
Figure 68 GUI UPF Diagram View



The GUI also provides a means to visualize the power state tables defined in the UPF. From the GUI menu bar, choose View > Toolbars > Power State Table to view any of the power state tables associated with the current block and better understand the relationship between the supplies available in the design. See Figure 69.

Feedback

Figure 69 GUI Power State Table



Placement and Routing Optimization

The placement, optimization, and clock tree synthesis steps in the IC Compiler II and Fusion Compiler tools take into account the voltage areas, power domains, always-on logic, power state table, and power management cells protecting the power domain interfaces. Unlike some other tools, the IC Compiler II and Fusion Compiler tools properly handle always-on logic without relying on always-on attributes of cells and nets.

Buffer Insertion

The IC Compiler II and Fusion Compiler tools dynamically evaluate buffer insertion based on the availability of always-on buffers in the reference libraries and the available compatible supplies. The tool simultaneously supports two different methods of always-on synthesis:

- Usage of dual-rail always-on buffers, which can be placed anywhere but require individual routing of backup supplies to the buffers. This method is very flexible but not economical due to area and congestion from the backup supply routing.
- Usage of ordinary single-rail buffers, which can be placed only in special-purpose, always-on row sites strategically located in the voltage areas. This method is economical when an always-on region is available nearby.

Because the tool supports both always-on methods, it can choose the best method for inserting each buffer to get an optimum balance between performance and cost.

Feedthrough Paths

Each new instance added during optimization or clock tree synthesis belongs to a voltage area, based on where the cell is placed. This voltage area is associated with a power domain, and the power domain is associated with a module in the logical hierarchy. As a result, by default, the tool inserts new cells in the logical hierarchy associated with the voltage area.

However, the tool can deviate from strict a logical-to-physical association using physical feedthrough optimization. A physical feedthrough path is a set of logical nets and cells that are physically placed in one voltage area but belong to a different module in the logical hierarchy.

For example, in a fully abutted floorplan, logical connections can exist between blocks that do not share a physical boundary. Typically, to allow connections of such logic, you manually add logical feedthrough ports to blocks not associated with the logical connections, based on the floorplan needs. This is a time-consuming process that requires an update each time the floorplan changes.

However, when you set the `opt.common.allow_physical_feedthrough` application option to `true`, the tool automatically uses physical feedthroughs for optimization (including high-fanout synthesis) and clock tree synthesis, which eliminates the need for manual logical feedthroughs port creation or port punching by the tool.

To keep the power domain information consistent with logical hierarchy of cells on these physical feedthrough paths, the tool assigns a new, separate power domain to the new cell instance on these feedthrough paths. The `save_upf` command writes out the new power domain mapping to the UPF-prime file or supplemental UPF file.

Note:

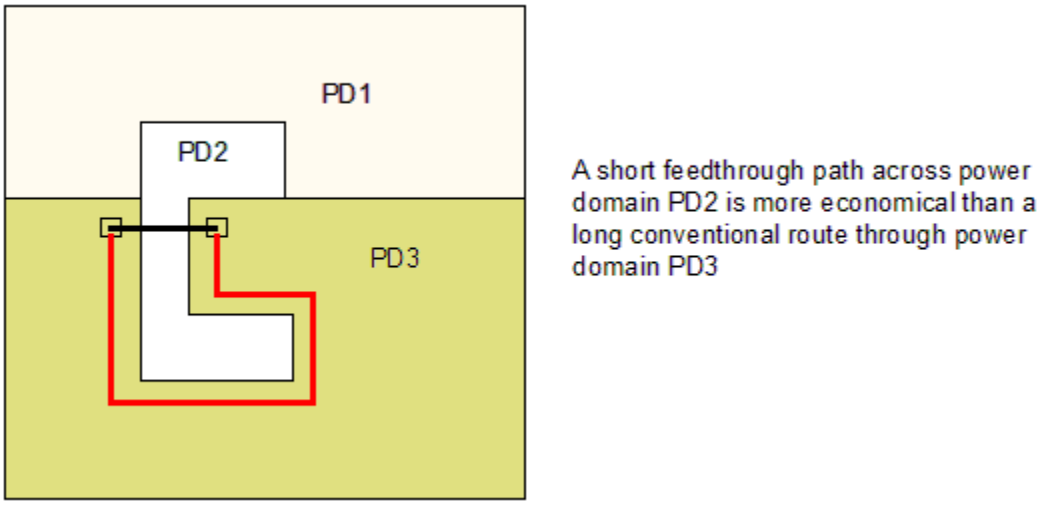
Some other tools support the usage of a separate power domain for feedthrough paths only if you create a wrapper for the feedthrough cells.

Routing

The IC Compiler II and Fusion Compiler router, Zroute, considers the multivoltage infrastructure and attempts to keep each route in the same voltage area and minimize the number of crossings over voltage area boundaries. This minimizes the need for always-on feedthrough connections. However, the router still creates feedthrough connections when the cost of detouring around a foreign voltage area is prohibitive. For example, it can be better to create a short feedthrough connection across a narrow foreign voltage area than to take a long route around it, as shown in the following figure.

Feedback

Figure 70 Feedthrough Path Example



Standard Rail Power and Ground Connections

Occasionally it might be necessary to connect power and ground nets by hand in the tool. To do this, use the `connect_supply_net` command:

```
prompt> connect_supply_net VSS -ports VSS
```

The `save_upf` command writes out these connections in either the UPF-prime file or supplemental UPF file.

For more information on creating power and ground routing, see the *IC Compiler II Design Planning User Guide* or *Fusion Compiler Design Planning User Guide*.

Saving the Design and ASCII Export

When the physical implementation is complete, you can save the block as a design view by using the `save_block` command. Saving the block stores all constraints, timing information, attributes, and application options.

You can export the design in ASCII format to use the netlist, constraints, and parasitic data with external verification tools or other tools. For correlation with PrimeTime, you can write out the SPEF, SDC, and Verilog netlist data. For example,

```
current_scenario Scenario1
write_parasitics -output myblock
write_verilog -exclude {all_physical_cells supply_statements} ./icc2.v
write_script -nosplit -format pt -output sdc_for_pt.tcl
save_upf myUPF.upf
save_block myblock
```

Feedback

In the Golden UPF flow (when the `mv.upf.enable_golden_upf` application option is set to `true`), the `save_upf` command writes out only the supplemental UPF file.

The recommended formats for exporting a UPF-specified design to the Formality tool for formal verification or the VC-LP tool for multivoltage checking is a Verilog netlist plus UPF files. The UPF files can be in either UPF-prime or golden UPF format.

Hierarchical Flow in IC Compiler II or Fusion Compiler With UPF

In the IC Compiler II or Fusion Compiler tool, hierarchical designs in all multivoltage UPF configurations can follow the same flow. The general handling of UPF data in the hierarchical flow is described in the following subsections. For details, see the *IC Compiler II Design Planning User Guide* or *Fusion Compiler Design Planning User Guide*.

Creating Block UPF From Full-Chip UPF

In the IC Compiler II or Fusion Compiler tool, to create the UPF for a lower-level block from the full-chip UPF, use the `split_constraints` command. This is the general procedure:

1. Load the full-chip netlist.
2. Load and commit the full-chip UPF.
3. Load the full-chip Synopsys design constraints (SDC).
4. Declare the lower-level blocks with the `set_budget_options -add_blocks ...` command.
5. Run the `split_constraints` command.

This automatically creates the UPF for the top-only block and each of the lower-level blocks, and also generates the SDC constraints for the same blocks.

Propagating Lower-Level Block UPF Into The Top Context

In the IC Compiler II or Fusion Compiler tool, block-level UPF is part of the block design data and is stored with the block. When you open a lower-level block with its UPF, the UPF is automatically loaded from the block into the top context. If you remove a lower-level block, the associated UPF is also removed. If you reload a modified version of the block, the corresponding UPF is automatically visible.

Writing Full Chip-UPF From Lower-Level Blocks With UPF

If you have opened several blocks that contain UPF and loaded a top-only UPF file, you can use the IC Compiler II or Fusion Compiler tool to write out an integrated flat UPF file for use in verification flows. Use the `save_upf` command with the `-full_chip` option:

```
prompt> save_upf -full_chip myFlat.upf
```

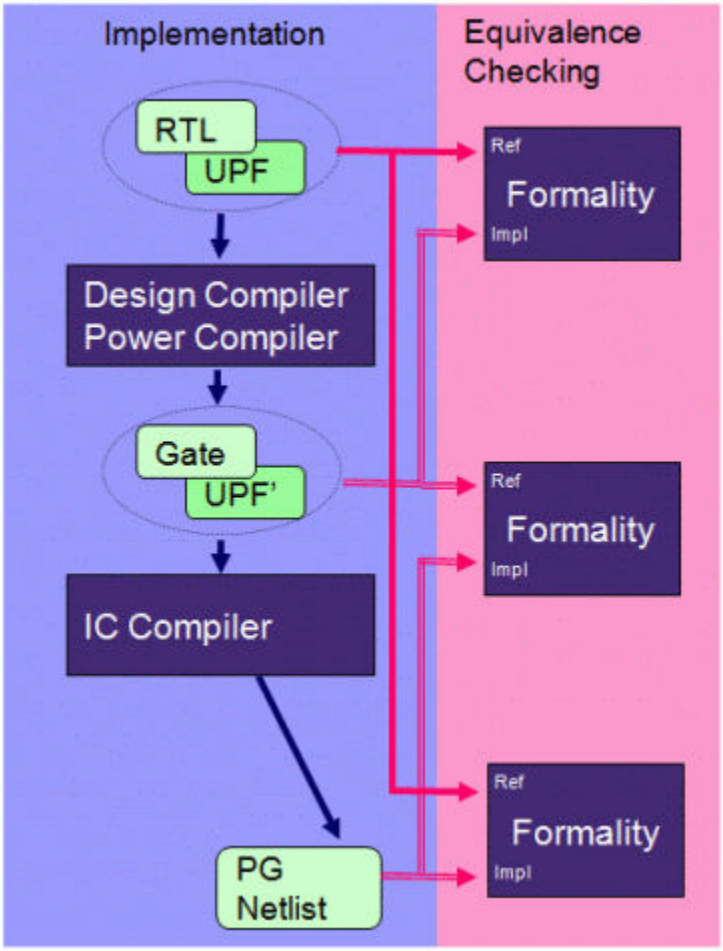
Feedback

Formal Verification Using Formality

Formality supports functional equivalence checking with low-power features such as clock gating, multiple-Vt, multivoltage supplies, power gating, and dynamic voltage and frequency scaling. Formality recognizes low-power cells such as isolation cells, level shifters, always-on cells, retention registers, and power gates.

Formality supports verification of low-power design data with UPF as shown in Figure 71.

Figure 71 Formality Equivalence Checking With UPF



The following types of design data comparison are supported:

- RTL + UPF versus Design Compiler gates + UPF'

  The RTL + UPF is the reference and the Design Compiler gate-level netlist (Verilog or .ddc format) and UPF' is the implementation.

Feedback

- RTL + UPF versus IC Compiler PG Connected Netlist

  The RTL + UPF is the reference and the IC Compiler PG-connected netlist (Verilog format) is the implementation.

- Design Compiler Gates + UPF' versus IC Compiler PG-connected netlist

  The Design Compiler gate-level netlist (Verilog or .ddc format) and UPF' are used for the reference and the IC Compiler PG-connected netlist (Verilog format) is used as the implementation.

Formal verification requires certain preparation steps with respect to libraries and the treatment of level shifters and isolation cells. To run Formality, all related logic libraries must be read in, meaning all logic libraries, including level shifter libraries. Library cells must have power pins and power-down functions defined. Before reading the reference and implementation designs, .svf (Formality guide) files created in the Design Compiler tool must be read by using the `set_svf` command. The .svf file contains information used in verification to improve performance and verification success.

Formality does not accept individual UPF commands entered interactively. The UPF description must be read into Formality with the `load_upf` command. You can use a single, top-level `load_upf` command, and for a hierarchical implementation flow, the top-level UPF script can contain additional `load_upf` commands.

The following script reads in RTL + UPF and a synthesized netlist + UPF' generated by the Design Compiler tool:

```
read_db {low_power_library.db special_lp_cells.db}
read_verilog -r { top.v block1.v block2.v block3.v }
set_top r:/WORK/top
load_upf -r top.upf
read_verilog -i { post_dc_netlist.v }
set_top i:/WORK/top
load_upf -i top_post_dc.upf
```

When you run the `load_upf` command, Formality checks the attributes of the library cells and issues warnings when the UPF file has missing or incomplete information. Here is an example of a summary reported generated by the `load_upf` command after checking the library:

```
************ Library Checking Summary ************
Warning: 54 unlinked power cell(s) with unread pg pins.
Warning: 6 linked power cell(s) with unread pg pins.
Warning: 74 unlinked power cell(s) with no power down functions on
outputs.
Error: 1 linked power cell(s) with no power down functions on outputs.
Warning: 389 unlinked power cell(s) with no power down function on an ff
 or latch.
Warning: 19 linked power cell(s) with no power down function on an ff or
 latch.
```

Feedback

```
   Use 'report_libraries -defects errors | all' for more details.
************************************************
```

To ensure accurate verification, you must correct these issues before you proceed. For Formality to automatically fix some of these defects, in the setup mode set the `hdlin_library_autocorrect` variable to `true`, before reading the UPF file.

Formality modifies the target reference and implementation designs to meet the specifications implied by the respective UPF command files. When a reference block is powered up, Formality verifies its functionality as usual. However, when a reference block is powered down, the compare points of the block are considered don’t care. Formality does not allow a reference X originating in an RTL+UPF “off” power domain to control compare points in an “on” power domain. Failing ports are reported if an X leaks out of a power domain in the implementation to downstream compare points, thus detecting the absence or malfunctioning of a needed isolation cell.

Formality recognizes and uses loaded UPF commands that create power domains, supply nets, supply ports, power switches, isolation cells, and retention registers. Because level shifters have no functional simulation effects, Formality does not insert level shifters that perform only buffering. Formality recognizes power state tables defined with the `create_pst` command.

Design Data Modification With UPF

Formality modifies the target design data (reference or implementation) to meet the specification implied by the UPF commands. In each type of comparison, Formality verifies the power-up functionality and reports failing points if the implementation netlist (or netlist plus UPF) powers down in a manner inconsistent with the reference design plus UPF. During verification, when an area in the reference design is powered down, Formality treats the compare points in this area as don't care.

In Formality, the `load_upf` command reads the file containing the UPF commands associated with the design data. It modifies the target design data to meet the specification implied by the UPF commands. As the RTL has been simulated and synthesized using a specific set of UPF commands, Formality must use those same commands to verify the RTL against the netlist. Therefore, UPF commands cannot be issued interactively in Formality. The `load_upf` command must be used after the `set_top` command after the design in the container has been successfully elaborated.

The following example shows part of a script using the `load_upf` command for the reference design:

```
read_verilog -r {top.v block1.v block2.v block3.v block1a.v}
set_top r:/WORK/top
load_upf -r top.upf
```

The `load_upf` command modifies the design data by adding nets and ports as implied by the `create_supply_net`, `create_supply_port`, and `connect_supply_net` UPF commands. It adds ports between hierarchical levels (performs “port punching”) as needed to make the supply connections. It also adds logic to the design based on the `create_power_switch`, `set_isolation`/`set_isolation_control`, and `set_retention`/`set_retention_control` UPF commands. After you execute `load_upf`, the design is modified and you can continue with the rest of the verification run.

Retention Registers

To allow formal verification of netlists synthesized from RTL, the behavior of the netlist must be consistent with the behavior of the RTL simulation. For any condition under which the RTL simulation value is not X for a register next state, primary output port, black box input pin, or other compare point, the value of the matching netlist compare point must be identical.

In the case of the UPF `set_retention` constructs, this level of consistency is not always possible for the following reasons:

- The Design Compiler tool chooses retention register implementations based on the `map_retention_cell` command.
- Actual retention register implementations demonstrate more complex and varied behavior than the UPF syntax can specify.

In many cases, the retention behavior of synthesized netlists is not expected to match the simulation behavior of the RTL from which it was synthesized, and is therefore not formally verifiable.

To achieve consistent RTL/netlist retention behavior and enable formal verification, use the `set_retention` and `set_retention_control` commands to define the retention strategies. These strategy definitions must match the retention behavior of the library cell specified in the `map_retention_cell` command. For example, you can use

```
set_retention -restore_condition {!clk}
```

to indicate that the clock signal must be low to restore the data. You must ensure that the corresponding library cell models are used in the `map_retention_cell` command while verifying the netlist against the RTL design.

Static Timing Analysis Using PrimeTime

The PrimeTime tool reads a gate-level netlist from the synthesis or physical implementation tool together with the UPF descriptions generated by those tools. It uses the UPF information to build a virtual model of the power network and to annotate voltage values on supply nets. This information allows calculation of appropriate voltage values on

Feedback

each power pin of each leaf-level gate instance in the design. You can explicitly specify the operating temperature of hierarchical cells. You can also override specific voltage values on individual power and ground pins of design cells.

PrimeTime reads and uses UPF information, but it does not modify the power domain description in any structural or functional way. Therefore, it does not write out any UPF commands with the `write_script` command and there is no `save_upf` command.

The following is a typical sequence of commands used in a PrimeTime timing analysis with UPF-specified power intent.

```
# Read libraries, designs
 ...
read_lib l1.db
read_verilog d1.v
 ...
link_design new_cpu

# Read UPF file

load_upf my_file.upf

# Define scaling library group
 ...
define_scaling_lib_group {lib_0.9V_0C.db lib_1.1V_0C.db lib_1.3V_0C.db}
...
set_voltage -cell ... -pg_pin_name ... value
# (sets voltage on supply nets or IR drop on power pins)
set_temperature -object_list cell_list value

# Read SDC and other timing assertions
read_sdc d1.tcl

# Read parasitics
read_parasitics my_rc.spef

# Perform timing, signal integrity analysis
report_timing
```

For power analysis using PrimePower, the following additional commands are typically used in the flow:

```
# Perform power analysis
# The most common flow alternatives are outlined in the following script:

# Time-based switching activity file
read_vcd file.vcd

# Control signal state probabilities
read_saif file.saif

# Set switching activity on signal nets
```

Feedback

```
set_switching_activity ...

# Set static values on control signal nets of the power switch
set_case_analysis ...

report_power
```

Voltage Scaling

The multivoltage flow can use either CCS timing or NLDM libraries. A set of CCS timing libraries that cover the range of voltages can be used with the voltage and temperature scaling capability in PrimeTime. Using NLDM libraries requires characterization at each of the voltages used in the design, and you set the `link_path_per_instance` variable to a list, with each list element consisting of a list of instances and the corresponding link paths that override the default link path for each of those instances.

With CCS timing libraries, PrimeTime supports voltage and temperature scaling by interpolating between data in separate libraries that have been characterized at different nominal voltage and temperature values. The delay (CCS timing driver model and receiver model) and timing constraints are scaled. In addition, scaling occurs if there is a mixture of CCS and NLDM data. Scaling between the libraries is done during runtime of the tool.You can invoke voltage and temperature scaling by using the `define_scaling_lib_group` command. This command specifies the scaling relationships between libraries that have been characterized at different voltages and temperatures and invokes both delay and constraint scaling, as shown in the following example:

```
pt_shell> define_scaling_lib_group \
          {lib_0.9V_0C.db lib_1.1V_0C.db lib_1.3V_0C.db}
```

This command should be issued after the design has been read in. If the design is not already linked, the `define_scaling_lib_group` command automatically links the design and creates scaling relationships between the libraries in each group. If the design is already linked, this command creates scaling relationships without an additional link. You can define multiple scaling groups to cover different portions of your design. However, each scaling group should contain the correct libraries for the type of scaling being performed (voltage, temperature, or voltage and temperature), and each library can be part of only one scaling group.

For more information about voltage and temperature scaling, see the section called “Scaling With CCS Timing Libraries” in the *PrimeTime User Guide*.

Setting Supply Voltages and Temperature

The `set_voltage` command specifies the voltage value on a supply net or PG pin of a cell, in volts. You can specify different minimum and maximum delay voltage values. You can use the `-dynamic` and `-min_dynamic` options to specify the dynamic portion

of the supply voltage (the portion that can vary across successive clock cycles). The `set_temperature` command specifies the operating temperature for a list of cells, in degrees Celsius. You can specify different minimum and maximum delay temperature values.

PrimeTime determines the cell rail voltage from the following sources of information, in order of increasing priority:

- `voltage_map`
- `set_voltage` on a power net
- `set_voltage` on a power pin
- design operating condition

The `voltage_map` statement in the library description of the cell (in Liberty format) defines the power supplies and default voltage values. The `related_power_pin`, `input_signal_level`, and `output_signal_level` attributes in the library cell description specify which power supply is connected to the pin's transistor stage. The voltage signal level margins are defined by the `input_voltage` and `output_voltage` attributes in the library description.

On-Chip Variation Analysis

In PrimeTime, the operating condition setting does not constrain the design linker. Note that the link is implemented before any SDC input. You can set any operating condition after the linking is complete. If CCS timing scaling library groups are used, the delay calculation is performed at the specified operating condition, as long as it is within the voltage and temperature range of the scaling group. If NLDM libraries are used, it is recommended to use `link_path_per_instance` to assign the appropriate library to an instance. Operating conditions beyond the range of the scaling library group should be avoided. If scaling library groups are not being used, it is recommended to use a library characterized at the required operating condition for both NLDM and CCS libraries.

On-chip variation analysis is recommended for timing sign-off. Best-case/worst-case analysis can produce results that are too optimistic in certain cases. In multivoltage designs, on-chip variation timing analysis can be performed one corner at a time using derate timing factors to model a slightly better or slightly worse condition around the main corner condition. To run this analysis, you must define a specific set of timing constraints to be targeted on only one corner.

When you use on-chip variation analysis, `set_voltage` *`max_case_voltage`* is used for maximum analysis and `set_voltage -min` *`min_case_voltage`* is used for minimum analysis. Note that with on-chip variation analysis, it is overly pessimistic to specify two different voltages for the same reason that it is overly pessimistic to specify two different best-case/worst-case operating conditions.

Feedback

Reporting and Checking Multivoltage Designs

Slew scaling is applied any time a signal transits across power domains without a level shifter. The `report_delay_calculation` command reports the slew scaling method used to take into account the different voltage levels between the driver and the load. For example,

```
pt_shell> report_delay_calculation -from I1/Z -to I2/B
  From pin: I1/Z
  To pin:   I2/B
  Main Library Units:  1ns  0.001pF  1000kOhm

  arc sense: unate
  arc type:  net

   RC network on pin 'I1/Z' :
   -------------------------------------------------
   Number of elements = 2 Capacitances + 1 Resistances
   Total capacitance  = 0.815687 pF
   Total capacitance  = 815.686687 (in library unit)
   Total resistance   = 8.323139 Kohm

   Scaling library pin group used for rise and fall.
   Scaling libraries used for receiver model : tc300c_0.85  tc300c_1.05

                               Rise        Fall
   ---------------------------------------------------------------------
   Net delay                 = 2.406542    2.836394  (in library unit)
   Transition time           = 5.312853    4.983530  (in library unit)
   From_pin transition time  = 0.804747    0.328441  (in library unit)
   To_pin transition time    = 5.312853    4.983530  (in library unit)
   Net slew degradation      = 4.508106    4.655089  (in library unit)
...
```

The `report_power_pin_info` command is useful for reporting the power pins of cells, including the voltage values. For example,

```
pt_shell> report_power_pin_info [get_cells -hierarchical]
...
Cell              Power Pin Name  Type           Voltage MaxD MinD  Power
 Net Connected
-------------------------------------------------------------------------
-------------
U61               VDDC            primary_power     1.1000  1.1000  VDD
U61               VSSC            primary_ground    0.0000  0.0000
 GND_main
do_reg_reg_5_     VDDC            primary_power     1.1000  1.1000  VDD
...
```

Feedback

To verify the signal-level consistency on the whole design, use the `check_timing` command. In the following example, the `check_timing` report shows some signal-level violations.

```
pt_shell> check_timing -include signal_level -verbose

Information: Checking 'unconstrained_endpoints'.
Information: Checking 'unexpandable_clocks'.
Information: Checking 'generic'.
Information: Checking 'latch_fanout'.
Information: Checking 'loops'.
Information: Checking 'generated_clocks'.
Information: Checking 'signal_level'.
Warning: There are 243 voltage mismatches MAX-MAX - driver rail !=
load rail:

Driver                  Voltage   Load                 Voltage   Margin
-------------------------------------------------------------------------
C_18_ASTlsInst72/OUT    1.08      Multiplier/CSA40/U492/A  1.32  -0.24
C_18_ASTlsInst72/OUT    1.08      Multiplier/CSA40/U656/A1 1.32  -0.24
C_18_ASTlsInst72/OUT    1.08      Multiplier/CSA40/U770/A  1.32  -0.24
Multiplier/S_26_/Q      1.32      GPRs/U7436/A             1.08  -0.24
```

To analyze the operating condition applied to the design paths during timing analysis, it is only possible to check the operating conditions applied on an instance-specific basis by using the `report_cell` command. For example,

```
pt_shell> report_cell ...
...
Cell              Reference     Library             Area  Attributes
-----------------------------------------------------------------------
S_reg_reg_27      DFFX2_QQT0    CORE_merge          25.65  n
-----------------------------------------------------------------------
Total 1 cell                                      25.65
```

PrimeTime SI supports static multivoltage crosstalk analysis, which deals with the voltage levels of the aggressor and victim nets. For more information, see the *PrimeTime User Guide*.

PrimePower Power Analysis

PrimePower, which is an extension of PrimeTime, performs comprehensive power analysis on gate-level designs. It can perform both average and peak power analysis and can generate detailed power reports.

PrimePower is built on the PrimeTime infrastructure. It uses the same commands and user interface as PrimeTime, and runs from the same PrimeTime shell. To enable PrimePower, set the variable `power_enable_analysis` to true:

```
pt_shell> set power_enable_analysis true
```

Feedback

To perform power analysis, the library (NLDM or CCS) must have power data tables. Also, switching activity data should be provided to the tool in the form of an SAIF file, VCD file, or `set_switching_activity` command. SAIF provides the toggle rate for average power analysis, whereas VCD provides data on all toggles for peak power analysis.

The following example is a PrimePower script for multivoltage designs:

```
## READ NETLIST

read_verilog top.v

## LINK THE DESIGN
set link_path "CORE_max_0v70_125c.db"
set link_path_per_instance [list [list {HV_INST} {* CORE_max_1v08.db}]]
lappend link_path_per_instance [list {HV_INST/U100} {* LS.db}]
link_design

## SOURCE UPF COMMANDS
load_upf ./core_upf.tcl

## SOURCE CONSTRAINTS
source ./cstr.tcl

## ANNOTATE PARASITICS
read_parasitics -keep_capacitive_coupling ./top.spef

## READ SWITCHING ACTIVITY
set_switching_activity [get_nets -hierarchical *] \
  -static_probability 0.1 -toggle_rate 0.1 -period 1
set_switching_activity [get_nets clock] \
  -static_probability 0.5 -toggle_rate 1.0 -period 1

## POWER ANALYSIS
set power_enable_analysis true

create_power_waveforms
report_power
```

The `report_power` command generates many useful power reports, such as peak power and average power reports, for every power domain and power net selected, for either the whole chip or a specific logic hierarchy. The following is an example of a power report that lists power consumption by different logic groups.

```
                   Internal  Switching  Leakage    Total
Power Group        Power     Power      Power      Power   (     %)
------------------------------------------------------------------------
io_pad                0.0000    0.0000    0.0000    0.0000 ( 0.00%)
memory                0.0000    0.0000    0.0000    0.0000 ( 0.00%)
black_box             0.0000    0.0000    0.0000    0.0000 ( 0.00%)
clock_network       4.688e-04 1.184e-04 5.424e-07 5.878e-04 ( 5.19%)  i
register            1.522e-03 3.473e-04 5.492e-07 1.870e-03 (16.52%)
combinational       3.843e-03 4.989e-03 2.637e-05 8.859e-03 (78.28%)
```

Feedback

```
sequential              0.0000    0.0000    0.0000    0.0000 ( 0.00%)

  Net Switching Power  = 5.455e-03   (48.20%)
  Cell Internal Power  = 5.834e-03   (51.55%)
  Cell Leakage Power   = 2.746e-05   ( 0.24%)
                         ---------
Total Power            =    0.0113  (100.00%)
```

For more information about the power analysis flow, see the *PrimePower User Guide*.

PrimeRail Power Network Analysis

PrimeRail extends the Synopsys sign-off solution for power supply network integrity checking. PrimeRail performs voltage drop and electromigration analysis for gate-level and transistor-level designs. It can be used for both static and dynamic power network analysis for a full-chip design.

For a multivoltage design, PrimeRail adds extra value by enabling dynamic power network analysis for a power-up sequence. The tool can also calculate the rush current during a power-up sequence.

Power Network Analysis Flow

The steps in the rail analysis flow depend on the library format used. A CCS power library stores leakage currents and dynamic current waveforms. Therefore, with CCS libraries, there is no need to characterize the library before running cell-level dynamic power and rail analysis. The tool reads the cell power characteristics directly from the CCS power library for transient power and rail analysis.

On the other hand, if NLDM libraries are used, a library should be characterized to capture the models of dynamic current waveforms and intrinsic parasitics using the HSPICE technology built into PrimeRail. While running rail analysis, the tool reads from these precharacterized models attached to the reference library.

To generate rail analysis maps and current waveforms, you should run a gate-level power analysis to generate the necessary timing and power information. For this purpose, PrimePower is made accessible through PrimeRail. After this step, PrimeRail reads the binary report of power consumption analysis to do rail analysis.

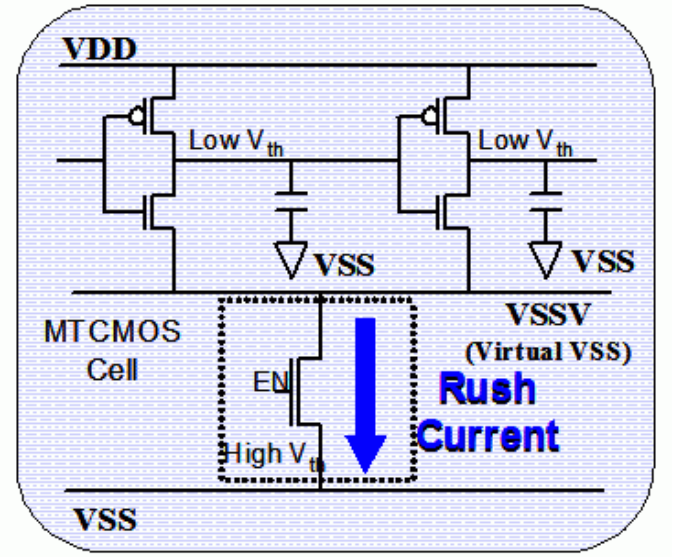
The rail analysis results can be viewed in the graphical user interface (GUI) in the form of a voltage drop map and waveforms. Also, you can generate reports for any voltage drop or current density violations.

Feedback

Power-Up Inrush Current Analysis

In the coarse-grain, multiple-threshold CMOS (MTCMOS) approach, power switch cells are used to cut off the power supply of an inactive power domain. The number of switch cells in the design can be large and turning them all on or off at the same time can draw a significant rush current from the power grid, as shown in Figure 72. As a designer, you must carefully control the power-up sequence of those power management transistors to minimize simultaneous switching noise when the circuits are turned.

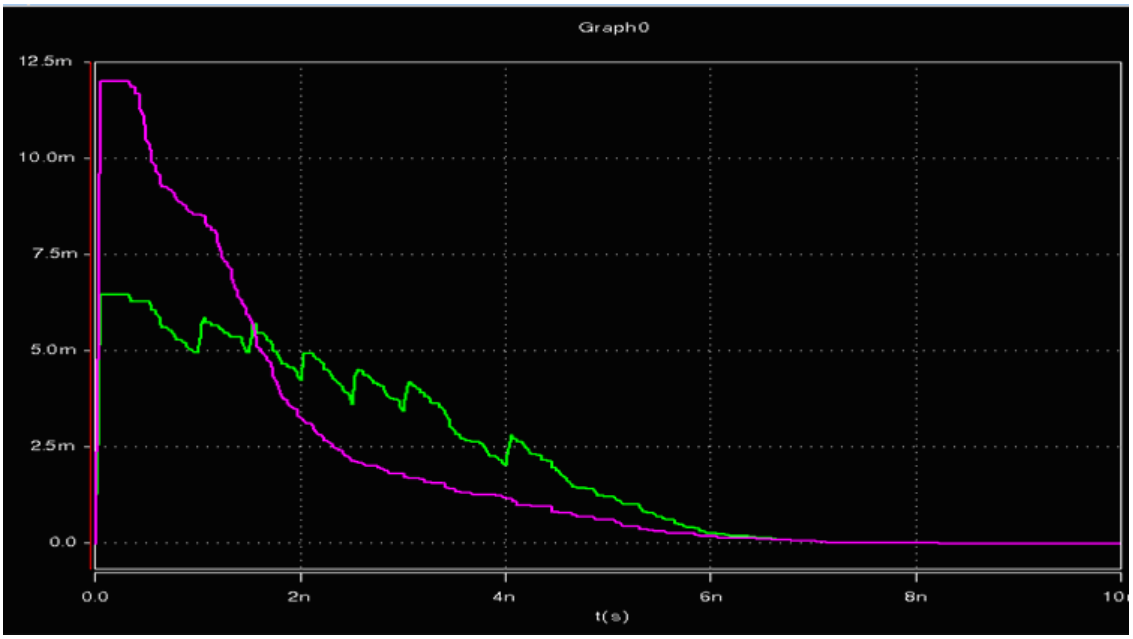
Figure 72 Rush Current Upon Power-Up



PrimeRail provides the capability to analyze the rush current effects of MTCMOS circuits. Figure 73 shows PrimeRail results for two MTCMOS implementations; one in which all the switch cells are powered up simultaneously, and the other in which switch cells are powered up sequentially. Note that a sequential implementation results in less peak rush current.

Feedback

Figure 73 Voltage Drop Waveform With Rush Currents



You can use PrimeRail results to optimize design parameters such as the number of power management cells, their drive strength, and the timing sequence of the control signals.

The flow for power-up sequence analysis is similar to power network analysis flow described previously. After extracting power and ground network, you can perform rush current analysis and wake-up time estimation. PrimeRail supports the display and reporting of rush current, virtual voltage waveforms, leakage current, and wake-up time.

For more information about power network analysis flow, see the *PrimeRail User Guide*.

Glossary

always-on
The characteristic of a cell, circuit, or power domain that is never powered down. For example, a logic cell used for isolation or retention is an always-on cell. The cell must have separate power supply pins for operation during power-down.

bubble register
The internal part of a retention register that is always supplied with power and retains data during power-down, constructed of higher-threshold transistors to minimize leakage current. Also called the shadow register.

clock gating
A method of reducing power by shutting off clocks to circuits that are not being used.

coarse-grain switching
A power-switching strategy that switches power on and off for a block as a whole, with all transistors in the block sharing a single power switch.

crowbar current
The current that flows from the power supply to ground current through a PMOS-NMOS stack during a logic transition, when both the PMOS and NMOS transistors are conducting for a brief period of time.

dynamic power
Power consumed by CMOS circuits during switching (changing of logic states), consisting of switching power and internal power.

dynamic voltage and frequency scaling
A speed-versus-power adjustment technique based on changing the supply voltage and operating frequency during operation to meet current workload requirements.

EDA
Electronic Design Automation, the process of using advanced software tools to design integrated circuits.

enable level shifter
A logic cell that performs both level-shifting and isolation functions, required where power domains can operate at different voltages and can be powered down.

extent
The set of cells in a design that belong to a power domain.

fine-grain switching
A power-switching strategy that switches power on and off for individual library cells, thereby allowing the power supply to each cell to be controlled individually.

footer switch

A power supply switch that connects the GND power supply to the GND supply pins of the switchable cells on the chip.

gate leakage

The current that flows between the gate and the source or drain of a transistor, caused by quantum-effect tunneling of electrons through the thin gate insulator.

header switch

A power supply switch that connects the chip's supply VDD to the VDD pins of the switchable cells on the chip.

internal power

Power consumed as a result of short-circuit (crowbar) current through a PMOS-NMOS stack during a logic transition, a type of dynamic power.

isolation cell

A logic cell that can isolate a power-down domain from a domain that is not powered down. The cell operates as a buffer when the input and output sides of the cell are both powered up, but provides a known, constant output signal when the input side is powered down.

level shifter

A circuit or library cell that converts signals from one voltage level to another in order to make the signal compatible with the supply voltage of the power domain at the output.

MTCMOS

Multiple-Threshold CMOS, a feature of a process technology that supports transistors having different threshold voltages. High-threshold transistors can be used where leakage reduction is important, while low-threshold transistors are used where switching speed is important.

multivoltage

The use of two or more different supply voltages on a chip.

NMOS

N-type Metal Oxide Semiconductor, a type of transistor consisting of an N-type source and N-type drain separated by a P-type channel. Applying a positive voltage on the gate induces an N-type conducting channel between the source and drain.

PG pin

A power or ground pin of a cell, as specified in the library definition of the cell.

PMOS

P-type Metal Oxide Semiconductor, a type of transistor consisting of a P-type source and P-type drain separated by an N-type channel. Applying a negative voltage on the gate induces a P-type conducting channel between the source and drain.

port-punching

The automatic creation of a power supply port by a synthesis or implementation tool, which makes a power connection from one hierarchical level to the next.

port state

A possible state of a power supply port. Each state has a name and an associated condition. The condition can be either a single voltage value, a set of three voltage values (minimum, nominal, and maximum), or the “off” state. The state names are used in power state tables.

power

The rate of energy usage, usually expressed in watts (joules per second) and calculated as current (amps) multiplied by voltage drop (volts).

power domain

A group of elements in the design that share a common power supply distribution network. The set of cells belonging to the power domain is called the extent of the power domain. All of these cells share the same set of power supply rails. The level of hierarchy where the power domain exists is called the scope of the power domain. A power domain must have one primary supply net and one primary ground net, and might optionally have additional supply nets, supply ports, and power switches.

power down

The process of shutting off the power to part of the chip by turning off a power switch.

power state table

A list of the allowed combinations of voltage values and states of the power switches for all power domains in the design.

power switch

A transistor or transistor array that can switch the power on and off for a portion of the chip, either at the VDD power supply (header switch) or at GND (footer switch), thereby cutting off power to portions of the chip during periods of inactivity. Conceptually, it is a device that turns on and turns off power for a supply net. A switch has an input supply net, an output supply net that can be switched on or off, and at least one input signal to control switching. The switch can optionally have multiple input control signals and one or more output acknowledge signals.

PST

Power state table; a list of the allowed combinations of voltage values and states of the power switches for all power domains in the design.

retention register

A memory register that retains data during power-down periods by keeping the data in a shadow register having an always-on power supply.

reused supply net

A supply net that spans different domains and passes through a supply port.

root cell

A cell defined as belonging to a power domain by the `-elements` option of the `create_power_domain` command, or specified as the scope of the command with the `-scope` option.

RTL

Register Transfer Level, a high-level description of a digital circuit expressed in terms of data transfers between registers and the logical operations performed on the data, written in a hardware description language such as Verilog or VHDL.

scope

The level of design hierarchy where a power domain exists.

shadow register

The internal part of a retention register that is always supplied with power and retains data during power-down, constructed of higher-threshold transistors to minimize leakage current. Also called the bubble register.

static power

The power consumed even when the circuit is not switching, consisting of reverse-biased p-n junction leakage, sub-threshold leakage, and gate leakage.

sub-threshold leakage

The small amount of current that flows between the source and drain of a transistor when the gate voltage is below what is considered the threshold voltage.

supply net

An abstract representation of a power or ground net in the design, representing a contiguous conductor that carries a supply voltage or ground connection.

supply port

A power supply connection point between adjacent levels of the design hierarchy. A supply net that crosses from one level of the design hierarchy to the next passes through a supply port.

supply set

An abstract representation of a bundle of supply nets that is specific to a power domain, that has a power and ground function definition.

supply set handle

An abstract supply set created for a power domain, which lets you synthesize a design even before you create any supply sets, supply nets, or supply ports for the domain.

switch

A power switch; a device that turns on and turns off power for a supply net.

switching activity

The relative presence or absence of logic transitions occurring on nets over a period of time, which affects the amount of dynamic power consumed over that period of time.

switching power

Power consumed as a result of charging and discharging the external capacitive load on the output of a cell.

threshold voltage

The gate-to-source voltage at which the transistor begins conducting between the source and drain.

UPF

Unified Power Format, a standard set of commands used to specify the low-power design intent for electronic systems, an alternative name of the IEEE 1801 Standard for Design and Verification of Low Power Integrated Circuits.

VDD

A name typically given to a power supply net or pin having a positive voltage.

voltage area

A physically contiguous area of the chip belonging to a single power domain and using the same power supply distribution network.

VSS

A name typically given to a ground-voltage supply net or pin.

Vt

The threshold voltage of a PMOS or NMOS transistor; the gate-to-source voltage at which the transistor begins conducting between the source and drain.