Synopsys® Multivoltage Implementation and Verification Overview

Version September 2024, September 2024

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

Feedback

About This Manual

An important part of the low-power digital implementation and verification flow is the support for the IEEE 1801 Standard for the 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 Synopsys Power Compiler™, Design Compiler®, VCS® LP, VC LP™, Formality®, IC Compiler™, IC Compiler™ II, Fusion Compiler™, PrimeTime®, and PrimePower.

This manual introduces and provides an overview of the multivoltage (UPF) flow using multiple Synopsys products, including synthesis, implementation, and verification products. It serves as a guide to the IEEE 1801 (UPF) usage in the 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 the individual Synopsys tools.

This manual is applicable for the following product versions:

Product	Version
Design Compiler, Formality, IC Compiler, IC Compiler II, Fusion Compiler, RTL Architect, Power Compiler, PrimePower, and PrimeTime	V-2023.12
VC LP and VCS NLP	U-2023.03-SP2

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

This preface includes the following sections:

New in This Release

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

To see the release notes for a product:

1. Go to the SolvNet Download Center.
2. Select the Synopsys product name and then, in the list that appears, select the product release version.
3. In the web page that appears, click **View release notes**.

Related Products, Publications, and Trademarks

For additional information about any of the tools mentioned in this guide, see their documentation on the Synopsys SolvNetPlus support site at the following address:

https://solvnetplus.synopsys.com

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

- Design Compiler® and Power Compiler™
- IC Compiler™
- IC Compiler™ II
- Fusion Compiler™
- RTL Architect™
- PrimeTime® and PrimePower
- Formality®
- PrimeRail
- VCS®
- Verdi®
- VC LP™

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`
`Purple`	• Within an example, indicates information of special interest. • Within a command-syntax section, indicates a default, such as `include_enclosing = true | false`
[ ]	Denotes optional arguments in syntax, such as `write_file [-format` *`fmt`*`ddc]`
...	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`
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Bold	Indicates a graphical user interface (GUI) element that has an action associated with it.
Edit > Copy	Indicates a path to a menu command, such as opening the **Edit** menu and choosing **Copy**.
Ctrl+C	Indicates a keyboard combination, such as holding down the Ctrl key and pressing C.

Customer Support

Customer support is available through SolvNetPlus.

Accessing SolvNetPlus

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

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

https://solvnetplus.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 SolvNetPlus site, click REGISTRATION HELP in the top-right menu bar.

Contacting Customer Support

To contact Customer Support, go to https://solvnetplus.synopsys.com.

Statement on Inclusivity and Diversity

Synopsys is committed to creating an inclusive environment where every employee, customer, and partner feels welcomed. We are reviewing and removing exclusionary language from our products and supporting customer-facing collateral. Our effort also includes internal initiatives to remove biased language from our engineering and working environment, including terms that are embedded in our software and IPs. At the same time, we are working to ensure that our web content and software applications are usable to people of varying abilities. You may still find examples of non-inclusive language in our software or documentation as our IPs implement industry-standard specifications that are currently under review to remove exclusionary language.

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.

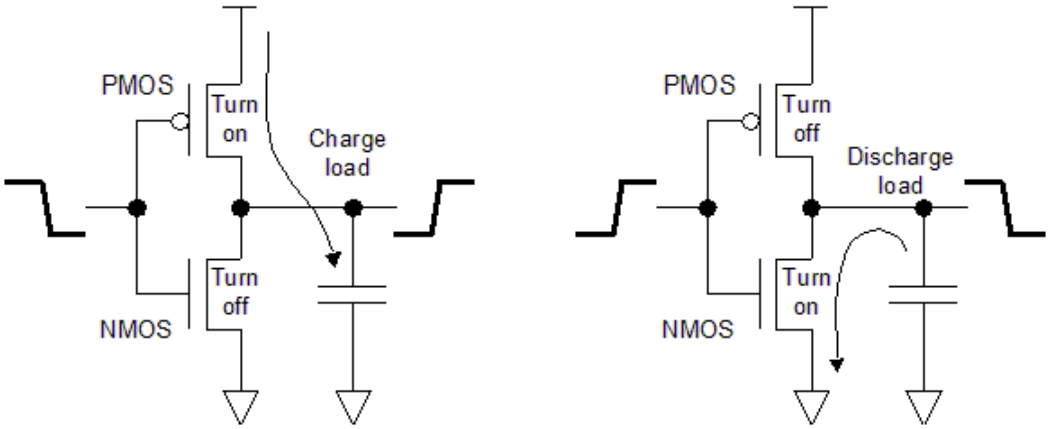
Feedback

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

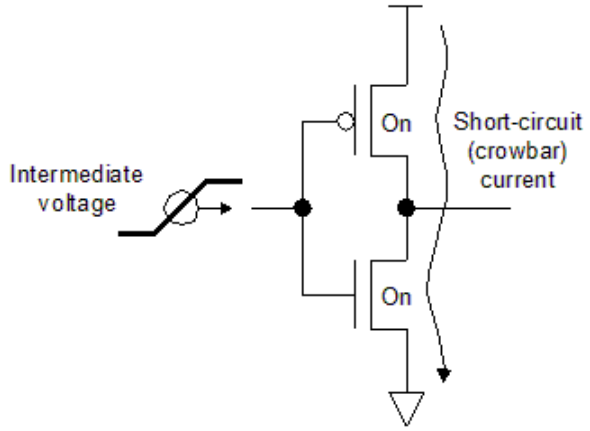


The energy used in each transition depends on the supply voltage and the capacitive load. Also, because the current flows only during logic transitions, the long-term dynamic power depends on the clock frequency and switching activity.

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. Higher threshold voltages and slower transitions result in more internal power.

Feedback

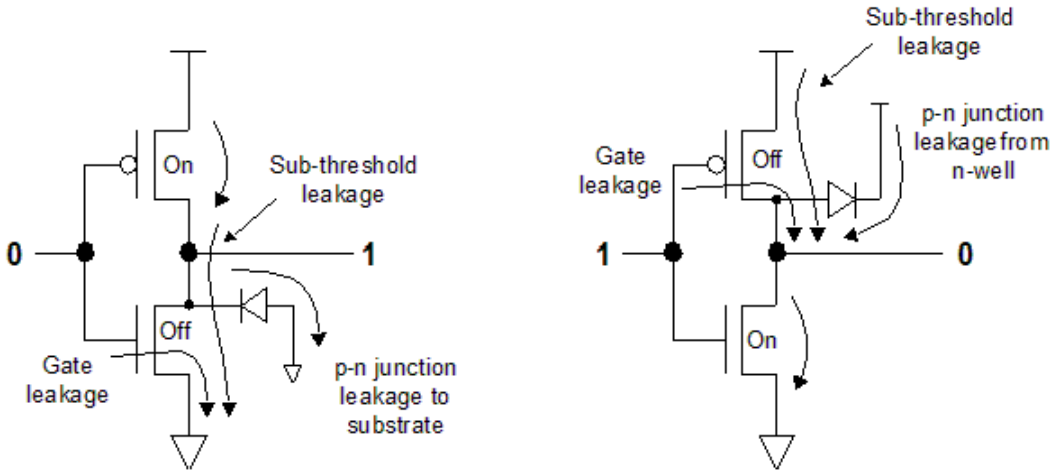
Figure 2 Internal Power



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.

Feedback

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.

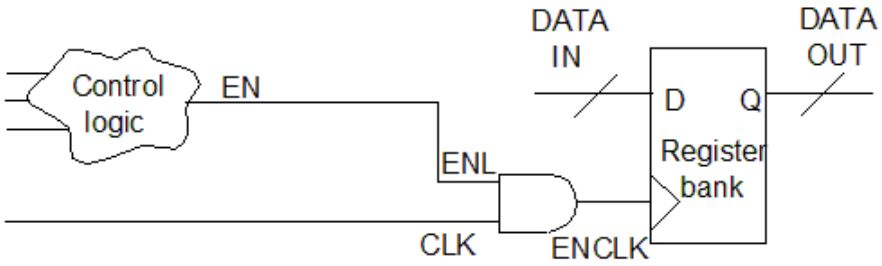
Each lowering of the supply voltage reduces per-gate power consumption, but also lowers the switching speed. In addition, the transistor threshold voltage must be lowered, causing more problems with noise immunity, crowbar currents, and sub-threshold leakage.

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

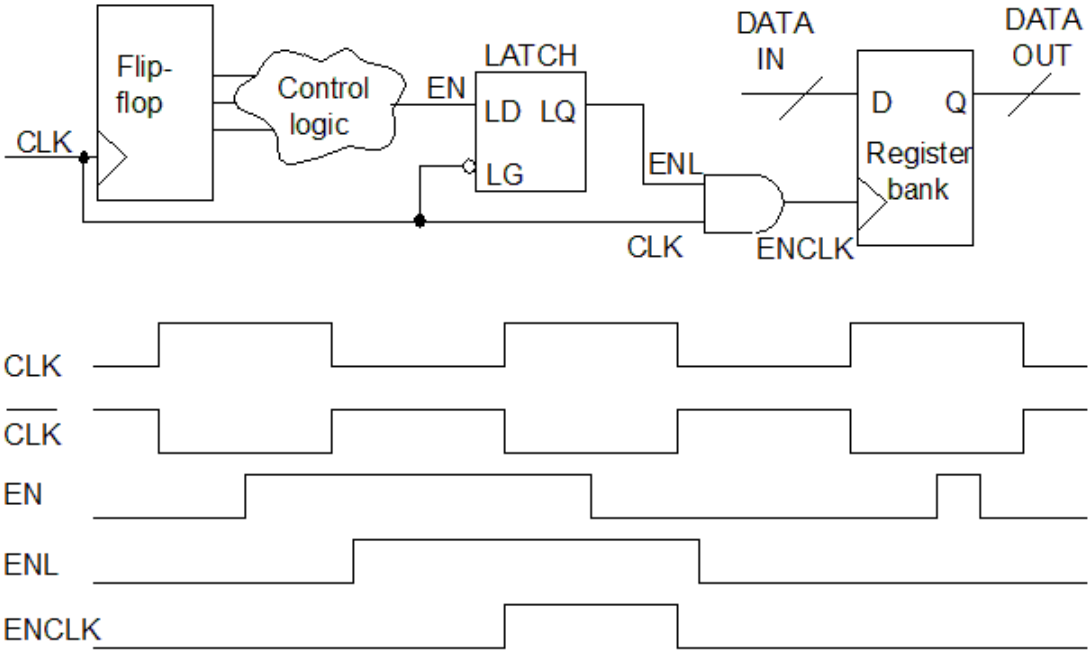


Clock gating is useful for registers that need to maintain the same logic values over many clock cycles. The main challenges are finding the best places to use it and creating the logic to shut off and turn on the clock at the proper times.

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



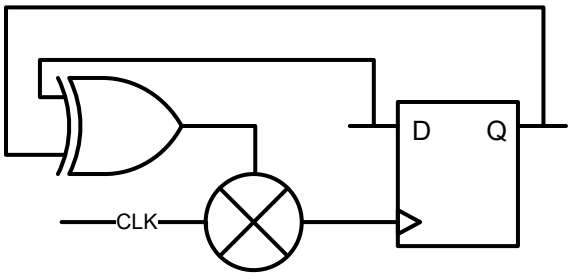
Feedback

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.

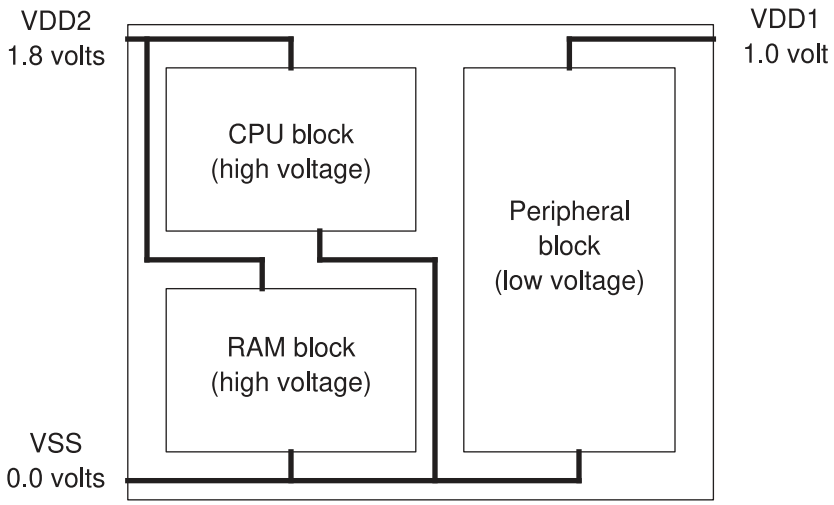
Multivoltage Design

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

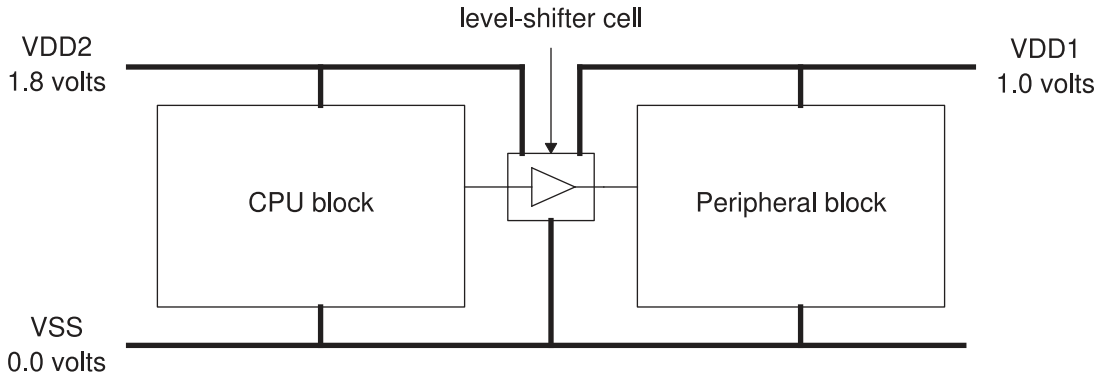


Providing multiple supply voltages on a single chip introduces some complexities and costs. Additional device pins must be available to supply the chip voltages, and the power grid must distribute each of the voltage supplies separately to the appropriate blocks.

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:

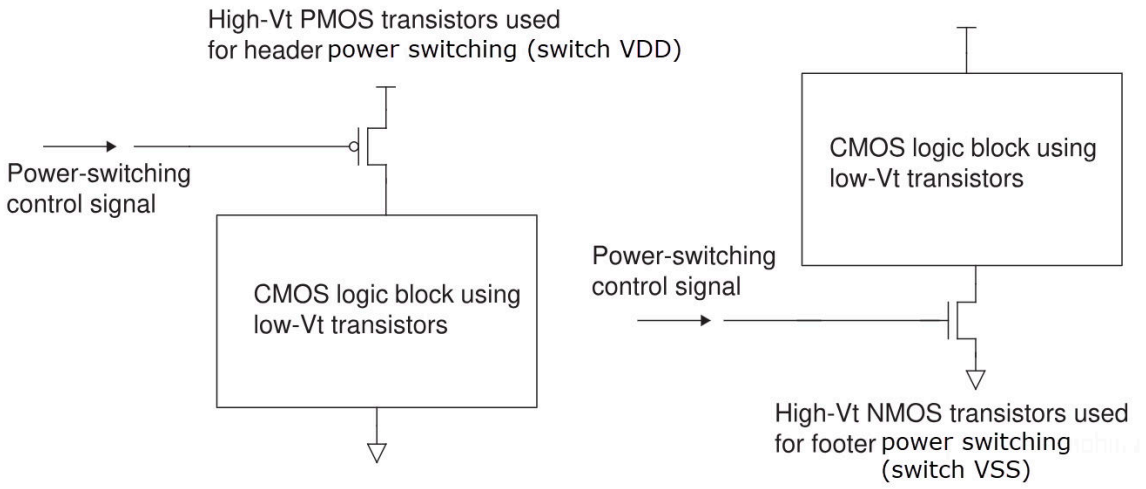
- Power controller – A logic block that determines when to power down and power up a specific block. There is some time and power cost for powering down and powering up a block, so the controller must determine the appropriate power-down times.
- Power switching network – A large number of high-Vt transistors 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.

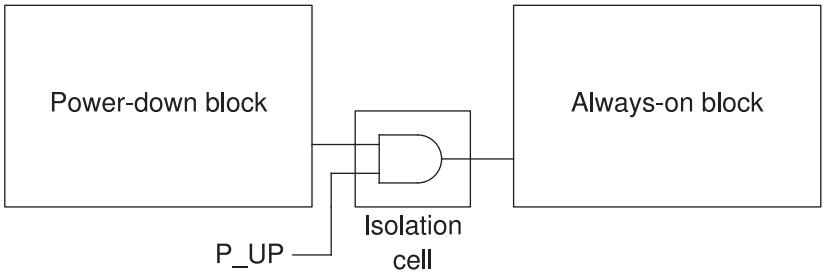
Isolation Implementation

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.

Feedback

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

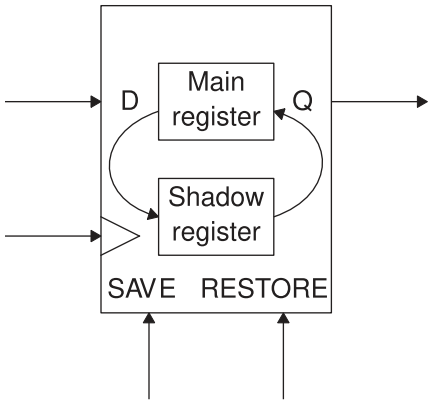


Retention Cell Implementation

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.

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 more information on the PG pin syntax and the modeling of power supply pin connections, see *Advanced Low-Power Modeling* in the *Library Compiler™ User Guide* in the Library Compiler Online Documentation.

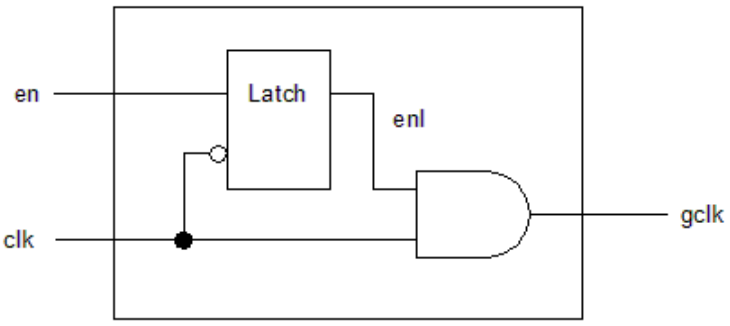
For an older library that does not have PG pins, you can quickly add PG pins with the `add_pg_pin_to_lib` or `add_pg_pin_to_db` command in Design Compiler or IC Compiler, thereby making the library compatible with the UPF power specification. Alternatively, you can use the `update_lib_model` command 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 on creating and using integrated clock-gating cells, see:

- *Modeling Power and Electromigration* in the *Library Compiler™ User Guide* in the Library Compiler Online Documentation
- *Clock Gating* in the *Power Compiler™ User Guide* in the Synthesis Online Help

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
```

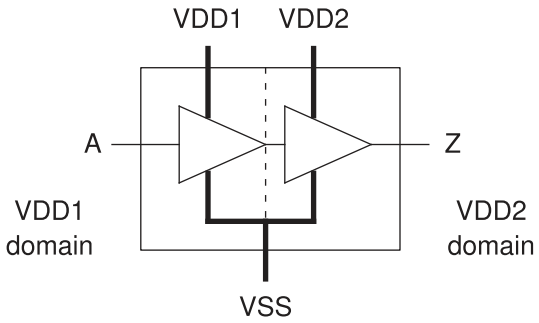
To set the multithreshold voltage constraint in the Design Compiler or IC Compiler tool, use the `set_multi_vth_constraint` command, or in the IC Compiler II or Fusion Compiler tool, use the `set_max_lvth_percentage` command. For details, see the man page for the command.

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);
    ...
  }
 ...
}
```

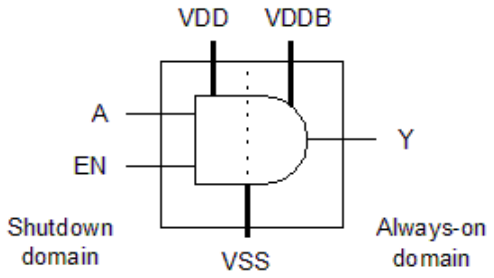
For more information on creating and using level-shifter cells, see:

- *Advanced Low-Power Modeling* in the *Library Compiler™ User Guide* in the Library Compiler Online Documentation
- *Specifying Level-Shifter Strategies* in the *Power Compiler™ User Guide* in the Synthesis Online Help

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.

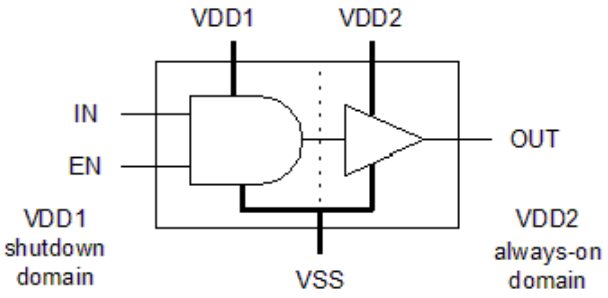
```
cell(Isolation_Cell) {
is_isolation_cell : true;
dont_touch : true;
dont_use : true;
pg_pin(VDD) {
  voltage_name : VDD;
```

```
    pg_type : primary_power;
  }
  pg_pin(VSS) {
    voltage_name : VSS;
    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.

Feedback

Figure 15 Enable Level Shifter



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;
}
...
}
```

Feedback

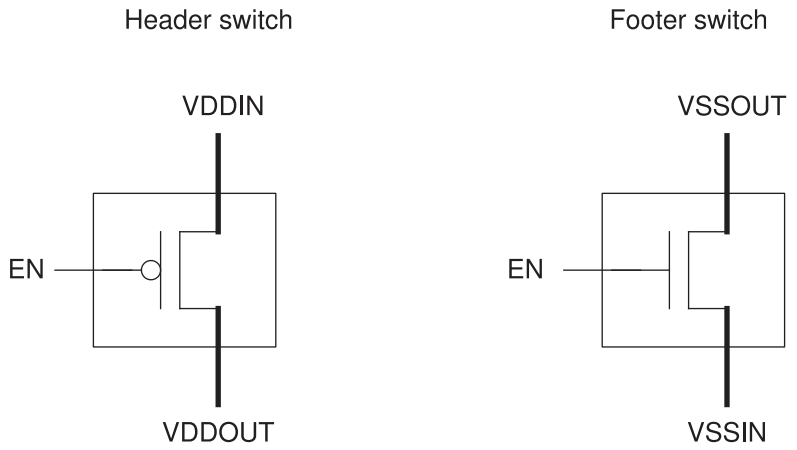
For more information on creating and using isolation and enable level-shifter cells, see:

- *Advanced Low-Power Modeling* in the *Library Compiler™ User Guide* in the Library Compiler Online Documentation
- *Specifying Isolation Strategies* in the *Power Compiler™ User Guide* in the Synthesis Online Help

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.

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 ) {
```

Feedback

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

For more information on creating power-switch cells, see *Advanced Low-Power Modeling* in the *Library Compiler™ User Guide* in the Library Compiler Online Documentation.

For information on using power-switch cells in physical planning, see:

- *IC Compiler™ Design Planning User Guide* in the IC Compiler Online Help
- *IC Compiler™ II Design Planning User Guide* in the IC Compiler II Online Help
- *Fusion Compiler™ Design Planning User Guide* in the Fusion Compiler Online Help

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.

Feedback

- 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.

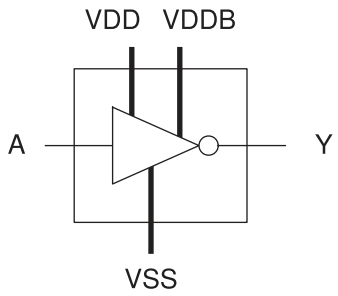
For more information on defining fine-grain switch cells, see *Advanced Low-Power Modeling* in the *Library Compiler™ User Guide* in the Library Compiler Online Documentation.

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.

Feedback

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

For more information on creating and using always-on logic cells, see:

- *Advanced Low-Power Modeling* in the *Library Compiler™ User Guide* in the Library Compiler Online Documentation
- *Always-On Logic* in the *Power Compiler™ User Guide* in the Synthesis Online Help

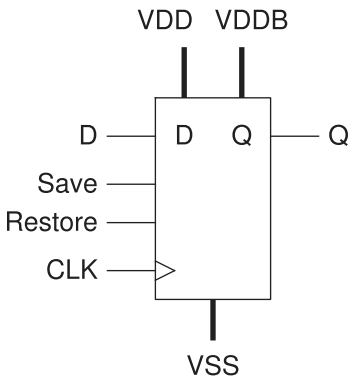
Feedback

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;
  }
  ...
  pin(RETN) {
    direction : input;
    capacitance : 1.0;
    nextstate_type : data ;
    related_power_pin :VDDB;
    related_ground_pin:VSSG;
    retention_pin (save_restore, "1" );
    }
```

Feedback

```
pin(Q) {
  power_down_function:"!VDD+VSS";
  related_power_pin : VDD ;
  related_ground_pin : VSS;
  direction : output;
  ...
```

For more information on retention register cells, see *Specifying Retention Strategies* in the *Power Compiler™ User Guide* in the Synthesis Online Help.

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 on macros with internal PG pins, see *Advanced Low-Power Modeling* in the *Library Compiler™ User Guide* in the Library Compiler Online Documentation.

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 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

Feedback

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

Note:

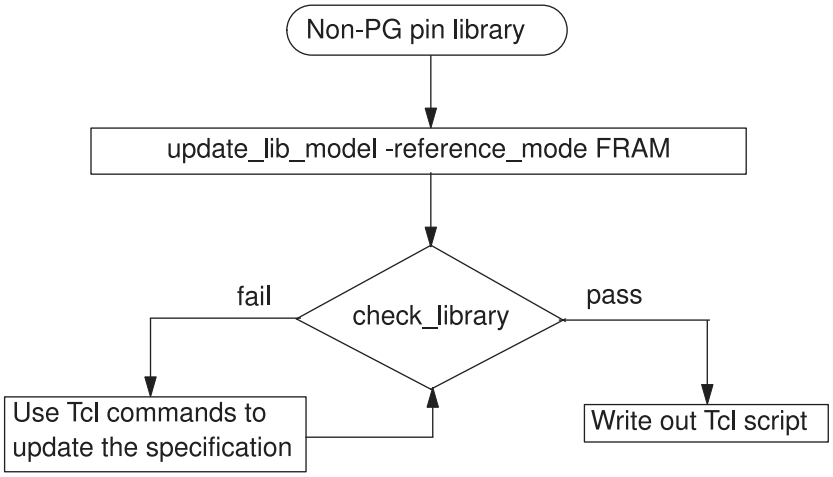
You must be logged in to SolvNetPlus for the link to connect directly to the article. If you are prompted to log in to SolvNetPlus upon clicking the link to the article, log in, then click the link again to reach the article.

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.

Feedback

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`* `...`

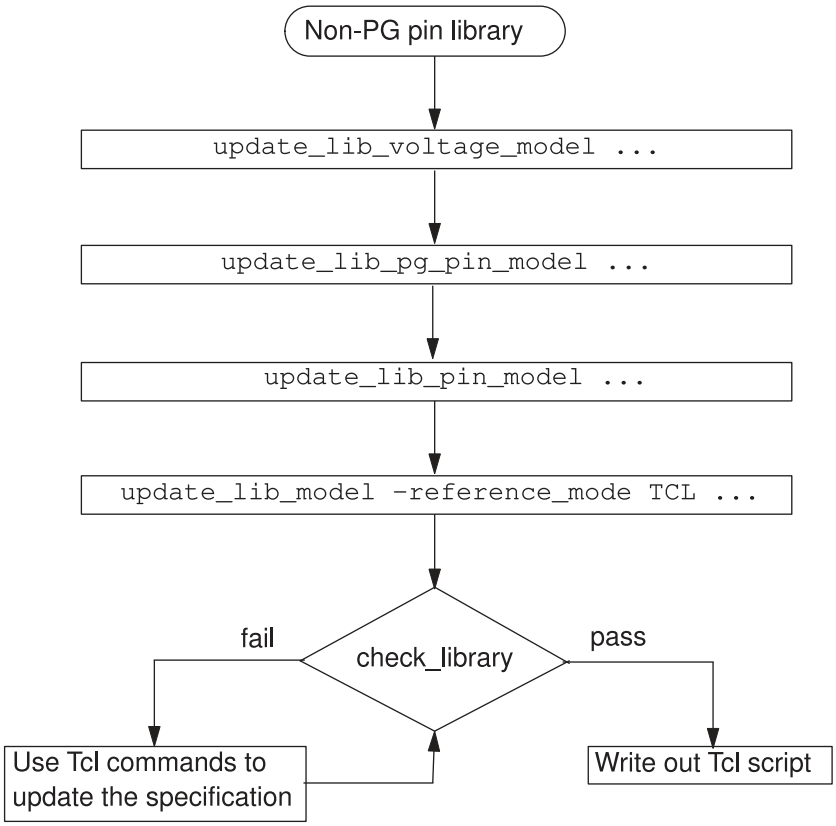
  This command sets the pin map for the specified library cell.

- `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 pages and *Library Checking* in *Library Quality Assurance System User Guide* in Library Compiler Online Documentation.

3

Power Intent Specification

The IEEE 1801 Standard for the 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 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 establishes a set of commands to specify the low-power design intent for electronic systems. Using UPF commands, you can specify the supply network, switches, isolation, retention, and other aspects relevant to power management of a chip design. The same set of low-power design specification commands is to be used throughout the design, analysis, verification, and implementation flow. Synopsys tools are designed to follow the official IEEE 1801 UPF standard.

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 for 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.

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 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.

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 level shifter 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*.

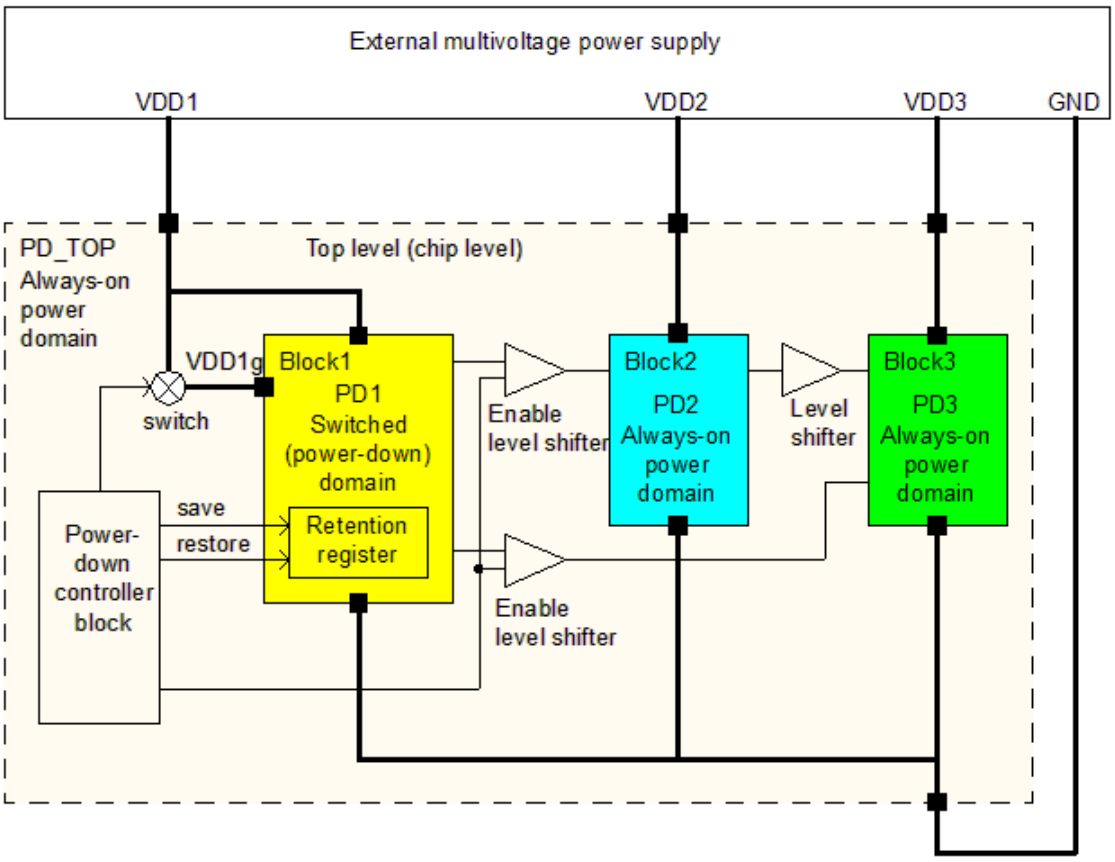
In a power domain that has power switching, any registers that must retain data during shutdown must be implemented as *retention registers*. A retention register has a separate, always-on supply net, sometimes called the backup supply, which keeps the data stable in the retention register while the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

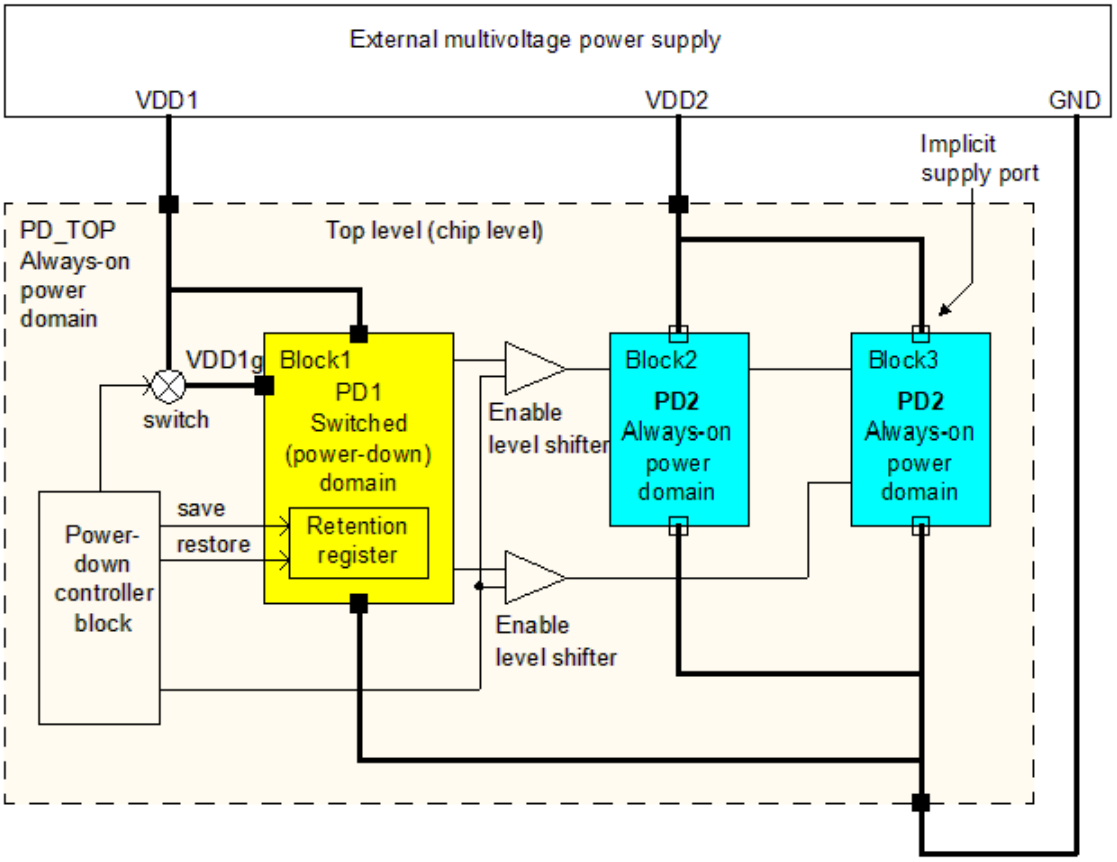
```
set_scope Block1
create_supply_net VDD1 -domain PD1
create_supply_port PRT1 -domain PD1
connect_supply_net VDD1 -ports {PRT1}
set_scope ...
```

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
```

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.

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.

See the following SolvNetPlus article for the complete list of UPF commands supported by the following tools: Design Compiler (Power Compiler), Fusion Compiler, IC Compiler II, RTL Architect, Formality, PrimeTime, PrimePower, VCS, and VC LP:

SolvNetPlus article 024016, “Synopsys Power Management: Supported Subset of the IEEE 1801 Unified Power Format (UPF) Standard.”

Note:

You must be logged in to SolvNetPlus for the link to connect directly to the article. If you are prompted to log in to SolvNetPlus upon clicking the link to the article, log in, then click the link again to reach the article.

Feedback

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.

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

Feedback

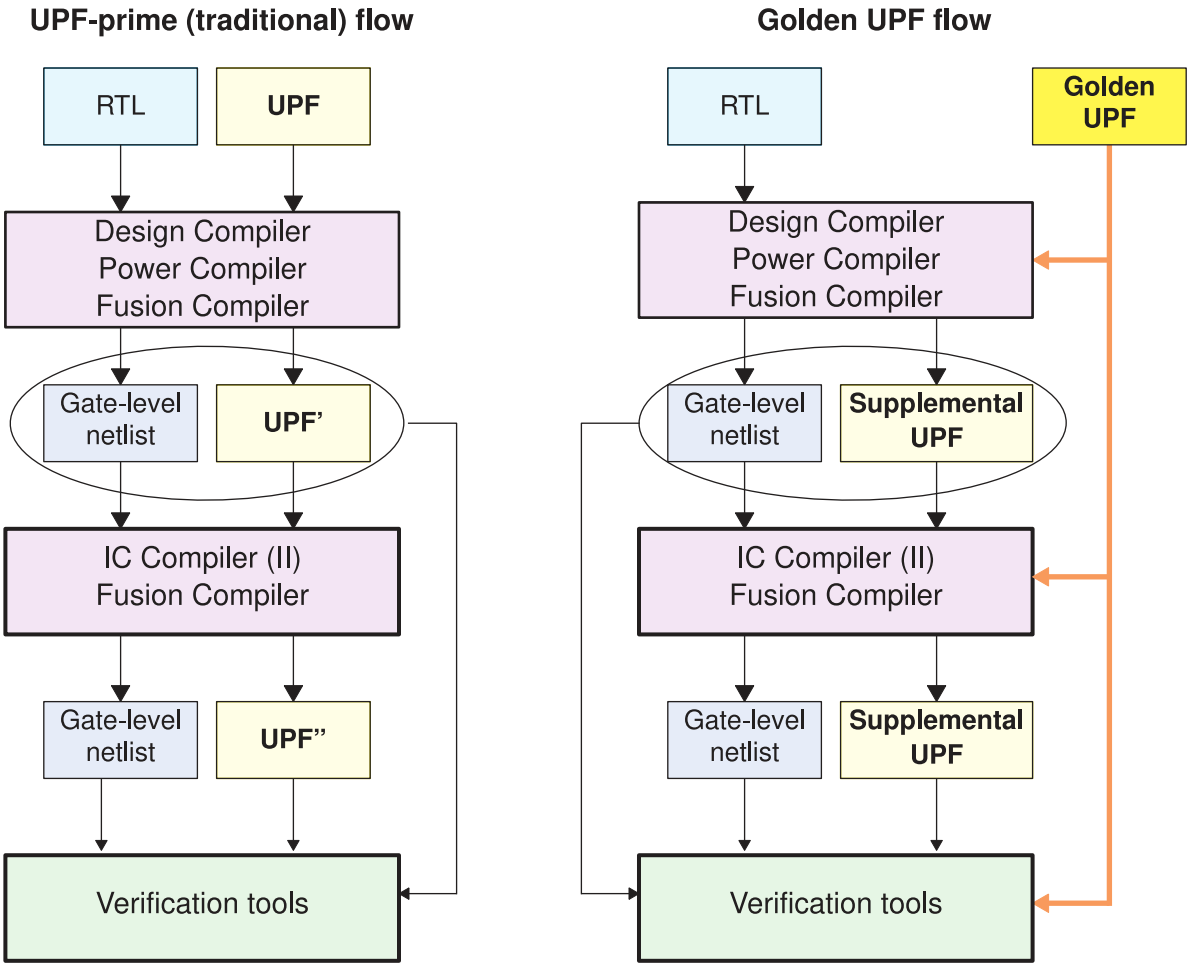
with the `save_upf` command, the tool writes out each of your entered commands as a comment line, immediately followed by the tool-modified command.

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, physical implementation, and verification steps, along with supplemental UPF files generated by the synthesis and physical implementation tools.

Figure 23 compares the traditional UPF-prime flow with the golden UPF flow.

Figure 23 UPF-Prime and Golden UPF Flows



Feedback

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.

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 SolvNetPlus article 1412864, “Golden UPF Flow Application Note.”

Note:

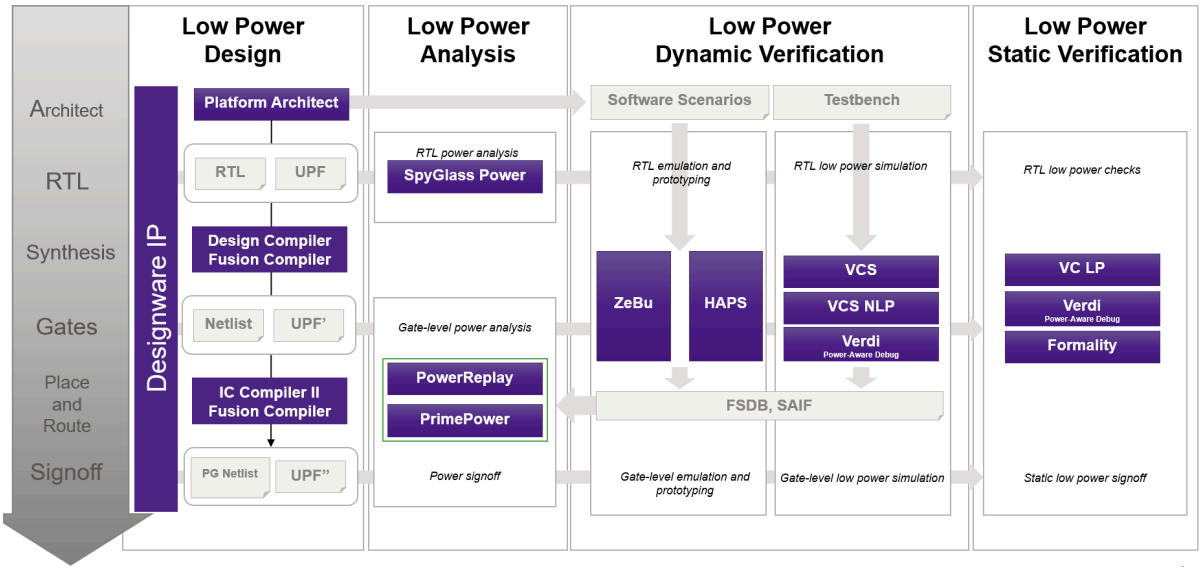
You must be logged in to SolvNetPlus for the link to connect directly to the article. If you are prompted to log in to SolvNetPlus upon clicking the link to the article, log in, then click the link again to reach the article.

4

Synopsys Multivoltage Flow

The Synopsys multivoltage synthesis, implementation, and verification flow is shown in Figure 24. 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.

Figure 24 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 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.

See also the following SolvNetPlus article for links to documentation related to the Synopsys multivoltage flow spanning multiple Synopsys tools ranging from synthesis to verification tools:

SolvNetPlus article 025518, "UPF and Multivoltage Flow Documentation References."

Note:

You must be logged in to SolvNetPlus for the link to connect directly to the article. If you are prompted to log in to SolvNetPlus upon clicking the link to the article, log in, then click the link again to reach the article.

5

UPF Script Examples

The following multivoltage flow examples demonstrate the UPF syntax used for specifying power intent.

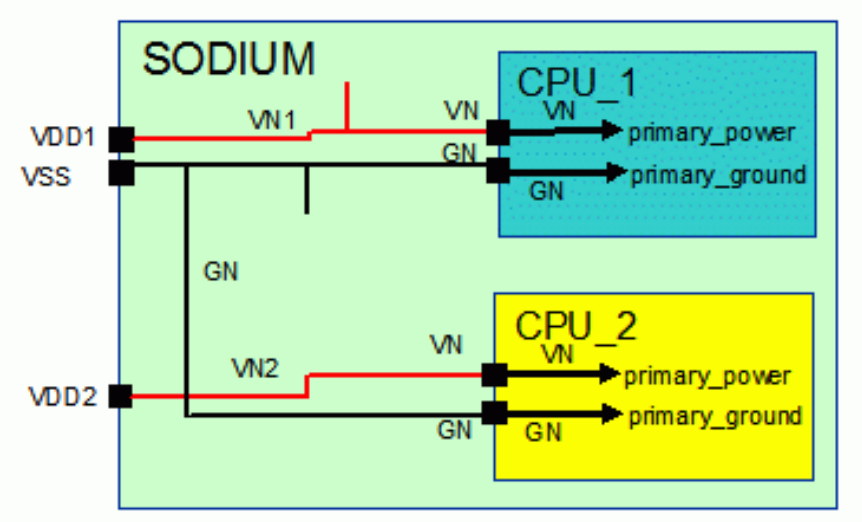
- Simple Multivoltage Design
- Supply Set Example
- Switched Power Supply Example
- Hierarchy and the get_supply_nets Command
- Power Switching States for a Macro Cell

Simple Multivoltage Design

The simple multivoltage chip design shown in Figure 25 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 25 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
```

Feedback

```
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} \
```

Feedback

```
  -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
```

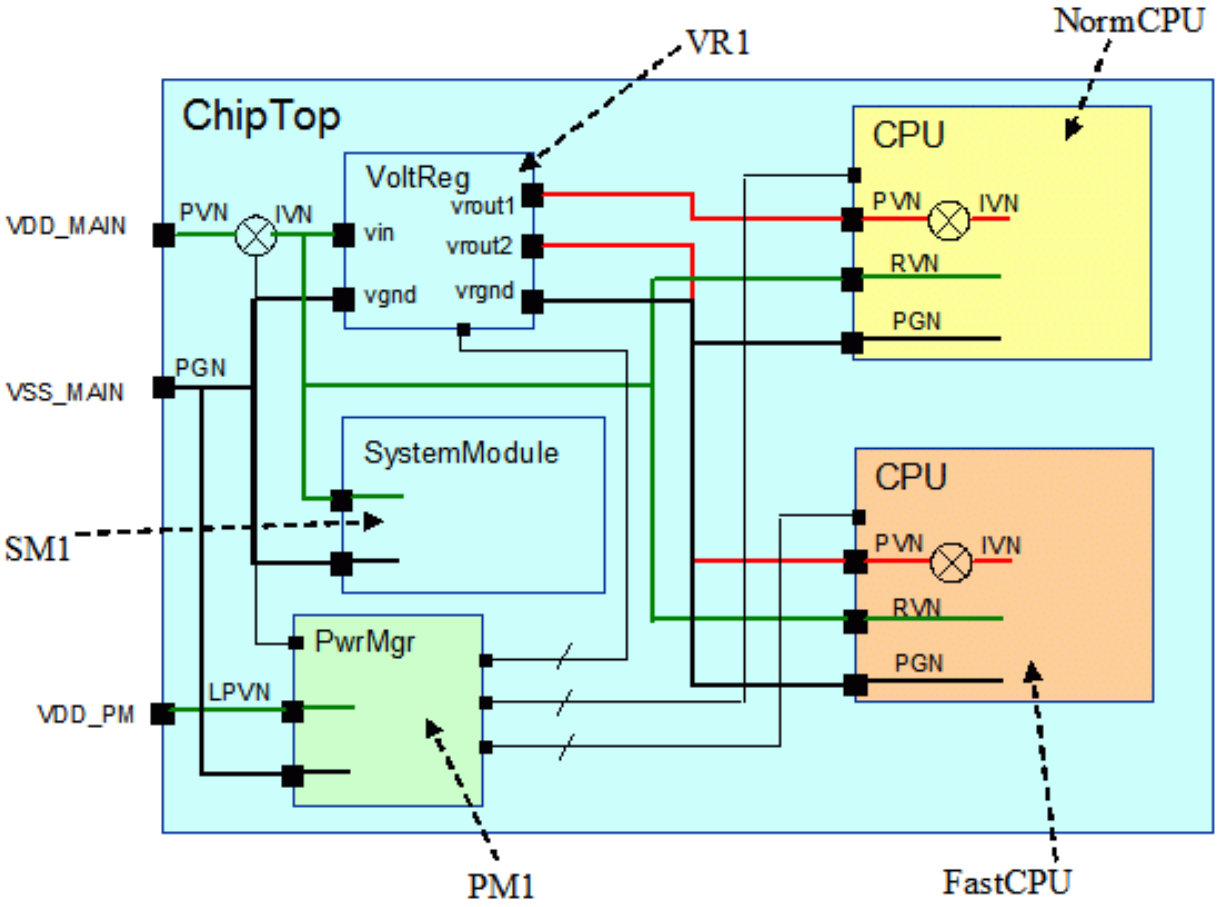
Feedback

```
set_port_attributes -elements {.} \
  -attribute related_supply_default_primary true
```

Switched Power Supply Example

The chip design shown in Figure 26 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 26 Simple Hierarchical Chip Design



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}
```

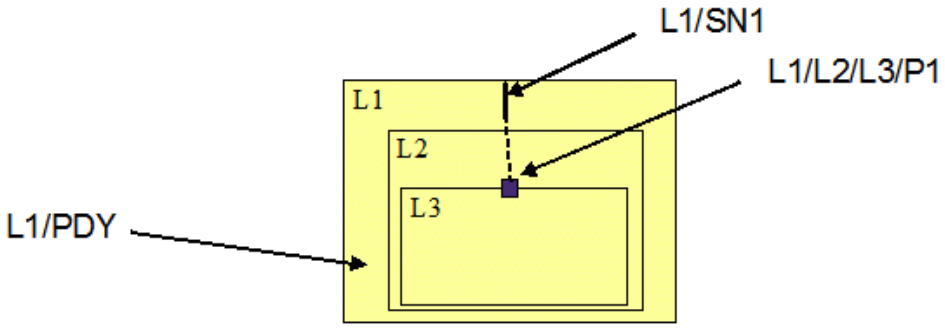
Feedback

```
# 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 27 and the following code example demonstrate how to use the `get_supply_nets` command to make a supply net connection across hierarchical levels.

Figure 27 Hierarchical Supply Connection Using get_supply_nets



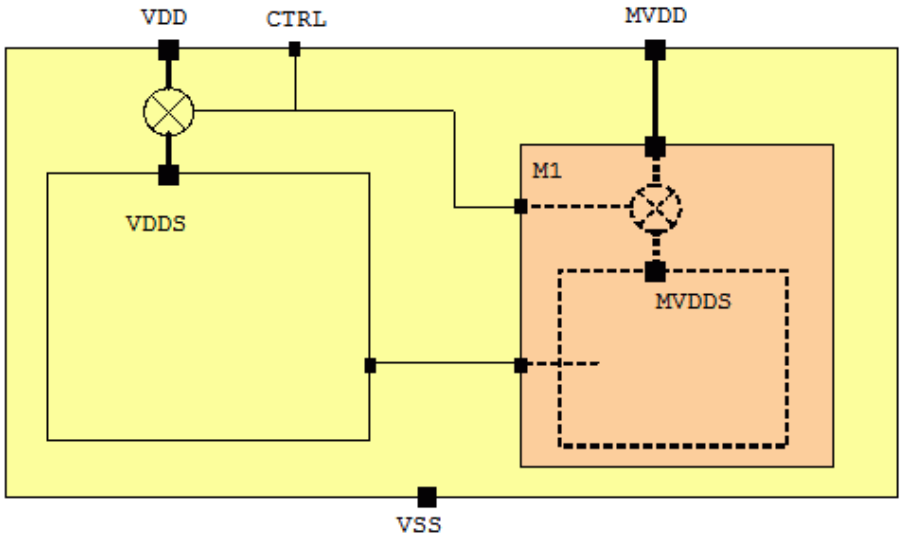

```
create_power_domain PDY -scope L1 -elements {L1}
create_supply_net SN1 -domain L1/PDY
create_supply_port L1/L2/L3/P1
set n [get_supply_nets SN1 -domain L1/PDY -scope L1/L2]
connect_supply_net $n -ports {L1/L2/L3/P1}
# an alternative to connect_supply_net L1/SN1 -ports {L1/L2/L3/P1}
```

Feedback

Power Switching States for a Macro Cell

In the power intent example shown in Figure 28, 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 28 Macro Cell With Power Switch

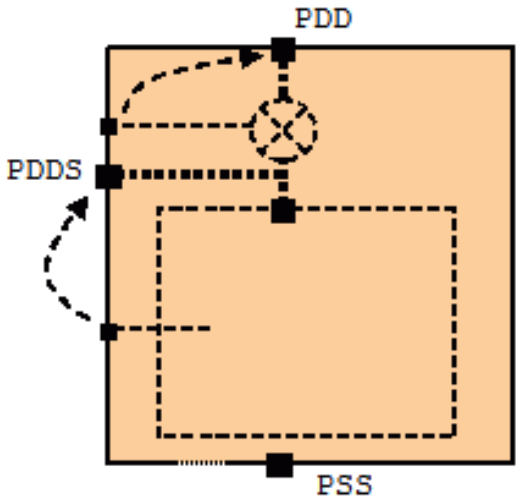


The design-level power switch is not yet available in the RTL. However, a UPF script is needed to create this power switch and describe the supply scheme of the design, such that all tools recognize that no isolation cell is needed between the macro cell and the rest of the circuit in the design.

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 29 schematically.

Feedback

Figure 29 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}
```

```
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
```

6

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

For more information, see the following topics:

- VCS NLP Native Low Power Simulation
- Debugging Low-Power Designs Using Verdi
- VC LP Static Low-Power Multivoltage Rule Checking
- Design Flow Stages and Multivoltage Checking

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.

For more information, see *Low Power Simulation* in the *VCS Native Low Power User Guide* in the VCS Online Documentation.

Debugging Low-Power Designs Using Verdi

The VCS NLP tool has the native Verdi integrated, which can be used for debugging low-power simulations. The Verdi tool 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, the VCS NLP tool instruments the low-power objects in HDL design to mimic the power intent specified in the UPF. The Verdi tool allows you to debug low-power objects instrumented by VCS NLP.

For more information, see *Debugging Low Power Designs* in the *VCS Native Low Power User Guide* in the VCS Online Documentation.

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, and 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.

For more information, see *VC LP Checking* in the *VC Static Platform User Guide* in the VC LP Online Help.

Design Flow Stages and Multivoltage Checking

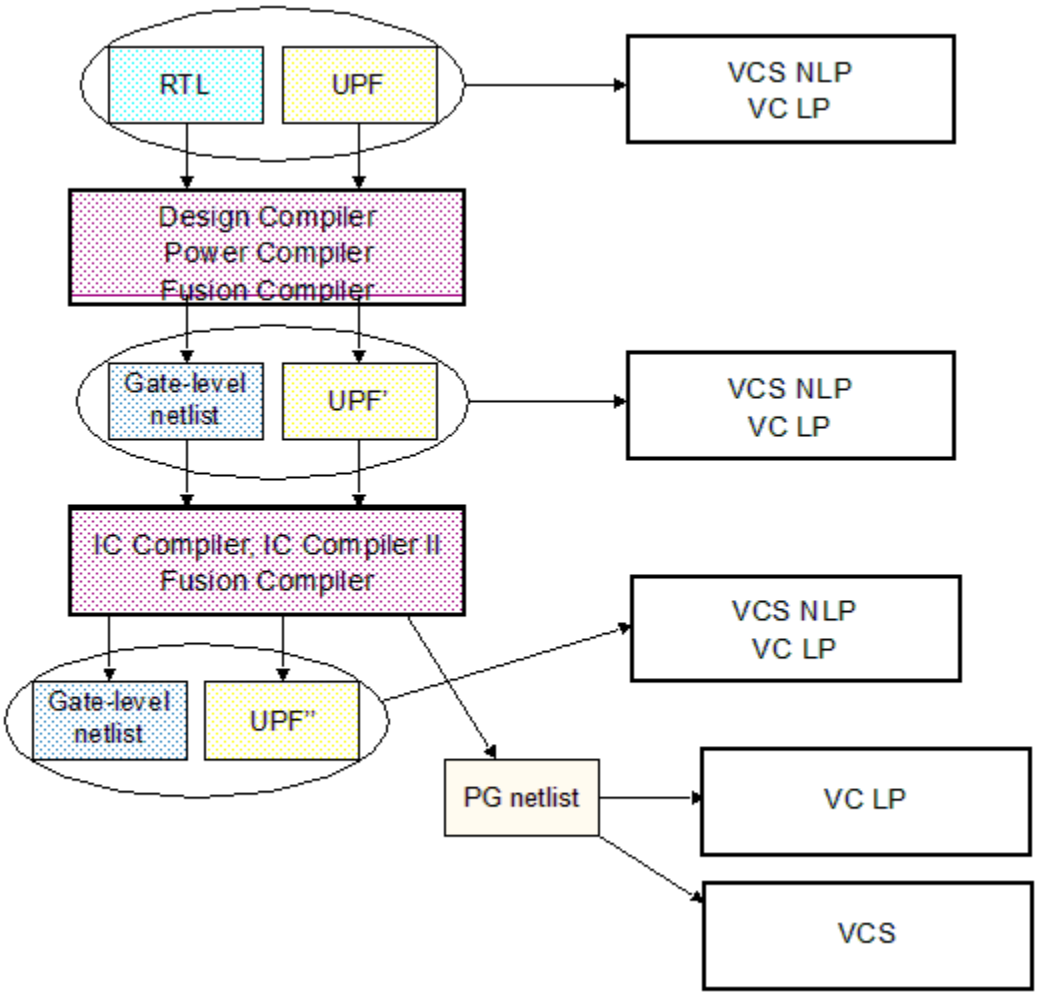
These are the stages in the design flow at which multivoltage simulation and rule checking can be performed:

- Original RTL + UPF, before synthesis
- Gate-level netlist + UPF’ produced by the Design Compiler and Power Compiler tools
- Gate-level netlist + UPF” and PG netlist produced by the IC Compiler, IC Compiler II, and Fusion Compiler tools

The types of checking that can be performed at each stage are summarized in Figure 30.

Feedback

Figure 30 Multivoltage Verification Flow



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.

Feedback

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.
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.

For more information, see the following topics:

- Power Domains and Hierarchy
- Specifying Operating Voltages
- Isolation, Level Shifter, and Retention Register Insertion
- Always-On Synthesis

- Compile
- DFT Methodology Using DFT Compiler

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 if 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.

For more information, see *Creating Power Domains* in the *Power Compiler™ User Guide* in the Synthesis Online Help.

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 more information, see *Multivoltage Design Flow Using UPF* in the *Power Compiler™ User Guide* in the Synthesis Online Help.

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.

For more information, see *Specifying Isolation Strategies*, *Specifying Level-Shifter Strategies*, *Specifying Retention Strategies*, and *Specifying Repeater Strategies* in the *Power Compiler™ User Guide* in the Synthesis Online Help.

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.

Feedback

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

For more information, see *Always-On Logic* in the *Power Compiler™ User Guide* in the Synthesis Online Help.

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:

The `compile_ultra` command, the default synthesis command in the 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, the compilation 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 the cell and pin levels to identify the retention cell and the "save" and "restore" control pins.

Feedback

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.

For more information on analyzing and reporting multivoltage aspects of a design, see *Examining and Debugging UPF Specifications* and *Reporting Commands for the UPF Flow* in the *Power Compiler™ User Guide* in the Synthesis Online Help.

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.

For more information, see the *Synopsys® TestMAX™ DFT User Guide* in the TestMAX DFT Online Help.

Logic Synthesis Using Fusion Compiler

The Fusion Compiler tool takes a design all the way from RTL to GDS. The Fusion Compiler `compile_fusion` command performs RTL synthesis and placement including mapping- and area-based optimization followed by logic-based delay optimization, placement and buffer-tree creation, and physical optimization for timing, power, and area. After `compile_fusion`, you execute the commands for clock tree synthesis (CTS) and post-CTS optimization, routing and post-route optimization, and signoff timing closure. The `compile_fusion` command unifies next generation synthesis and the placement and optimization steps of the IC Compiler II tool.

Feedback

In the Fusion Compiler synthesis flow for multivoltage designs, the `compile_fusion` command is used to insert mapped power management cells and perform an always-on synthesis. For detailed information on the flow diagram and the steps in the flow, see *Multivoltage Design Overview* in the *Fusion Compiler™ Multivoltage User Guide* in the Fusion Compiler Online Help.

The Fusion Compiler tool supports both the traditional UPF flow and the golden UPF flow.

In the traditional flow, also known as the UPF prime (UPF') flow, the RTL file containing the design intent and the UPF file containing the power intent are both read in by the Fusion Compiler tool. During synthesis, the tool might insert special cells such as isolation or level-shifter cells into the design to fulfill the UPF power intent. When logic synthesis is complete, the tool writes out a gated netlist and a UPF prime (UPF') file. The UPF' file contains any changes to the power intent that occurred during synthesis. The UPF' file is a mixture of original UPF commands and new or modified commands.

The golden UPF flow maintains and uses the original UPF file throughout the flow. The Fusion Compiler tool writes power intent changes into a separate supplemental UPF file. Downstream tools and verification tools then use the combination of the golden UPF file and the supplemental UPF file. For detailed information on the two UPF flows and their comparison, see *UPF Flows* in the *Fusion Compiler™ Multivoltage User Guide* in the Fusion Compiler Online Help.

Design Planning Using IC Compiler II or Fusion Compiler

You can use the IC Compiler II or the Fusion Compiler physical implementation tools for the 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.

For more information, see the following topics:

- Power Domains and Voltage Areas
- Power Management Cell Placement
- Power Planning

Feedback

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.

Both rectangular and rectilinear shapes are supported for voltage areas. 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.

The IC Compiler II 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.

For more information on creating voltage areas and assigning cells belonging to a specific power domain to the voltage area, see:

- *IC Compiler™ II Design Planning User Guide* in the IC Compiler II Online Help.
- *IC Compiler™ II Multivoltage User Guide* in the IC Compiler II Online Help.
- *Fusion Compiler™ Design Planning User Guide* in the Fusion Compiler Online Help.
- *Fusion Compiler™ Multivoltage User Guide* in the Fusion Compiler Online Help.

Power Management Cell Placement

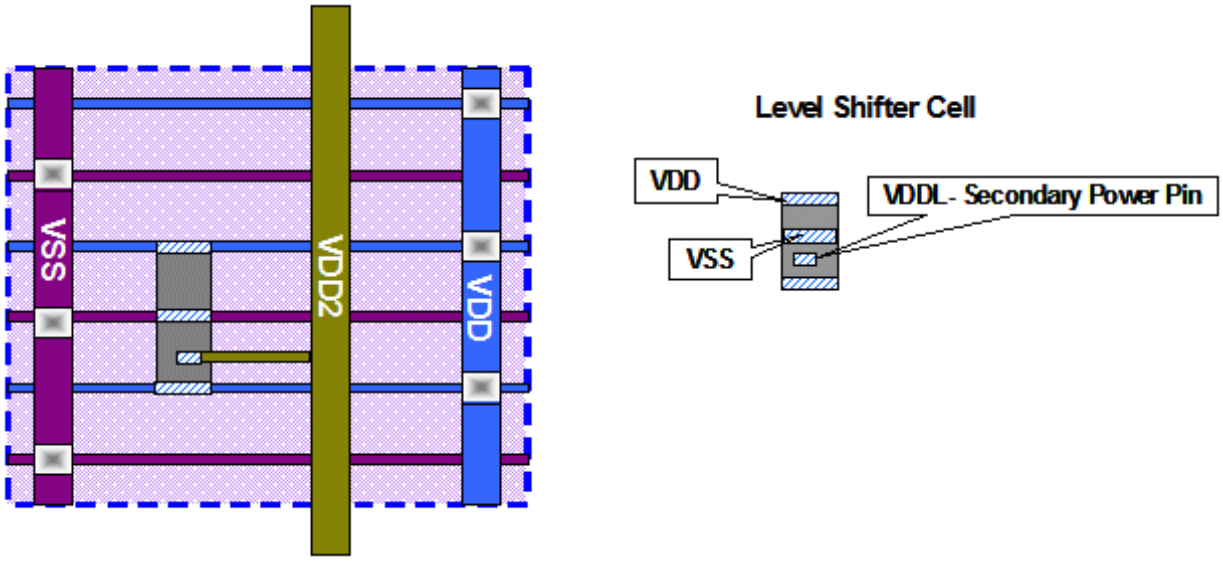
Power-related cells such as level shifters, isolation cells, retention registers, always-on cells, and power switches typically have multiple power and ground pins and are taller

Feedback

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 31. The cell is taller than a standard cell and has 2X height.

Figure 31 Level Shifter With Dual Power Pins



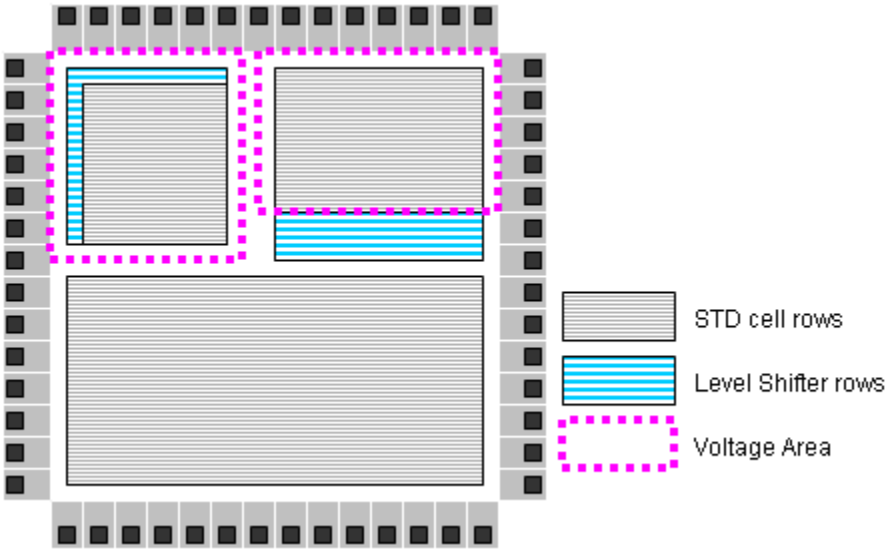
The secondary power pin, VDDL, should be connected to the power supply net VDD2. For power supply routing, VDDL needs to be routed to the VDD2 strap. However, this cannot be achieved with the normal signal or power routing method. A special type of routing, called net mode routing (specified with the 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 31 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 32 shows an example of what separate level-shifter placement areas might look like.

Feedback

Figure 32 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.

For more information on implementing power switches, see:

- *Power-Switch Cells* in the *IC Compiler™ II Multivoltage User Guide* in the IC Compiler II Online Help.
- *Power-Switch Cells* in the *Fusion Compiler™ Multivoltage User Guide* in the Fusion Compiler Online Help.

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.

For more information about power management cells and always-on logic, and their implementation, see:

- *Power Management Cells* in the *IC Compiler™ II Multivoltage User Guide* in the IC Compiler II Online Help.
- *Power Management Cells* in the *Fusion Compiler™ Multivoltage User Guide* in the Fusion Compiler Online Help.

Power Planning

After you complete the design planning process and have a complete floorplan, you can perform power planning.

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.

For more information on power planning, including power network synthesis and power network analysis, see:

- *Performing Power Planning* in the *IC Compiler™ Design Planning User Guide* in the IC Compiler II Online Help.
- *Performing Power Planning* in the *Fusion Compiler™ Design Planning User Guide* in the Fusion Compiler Online Help.

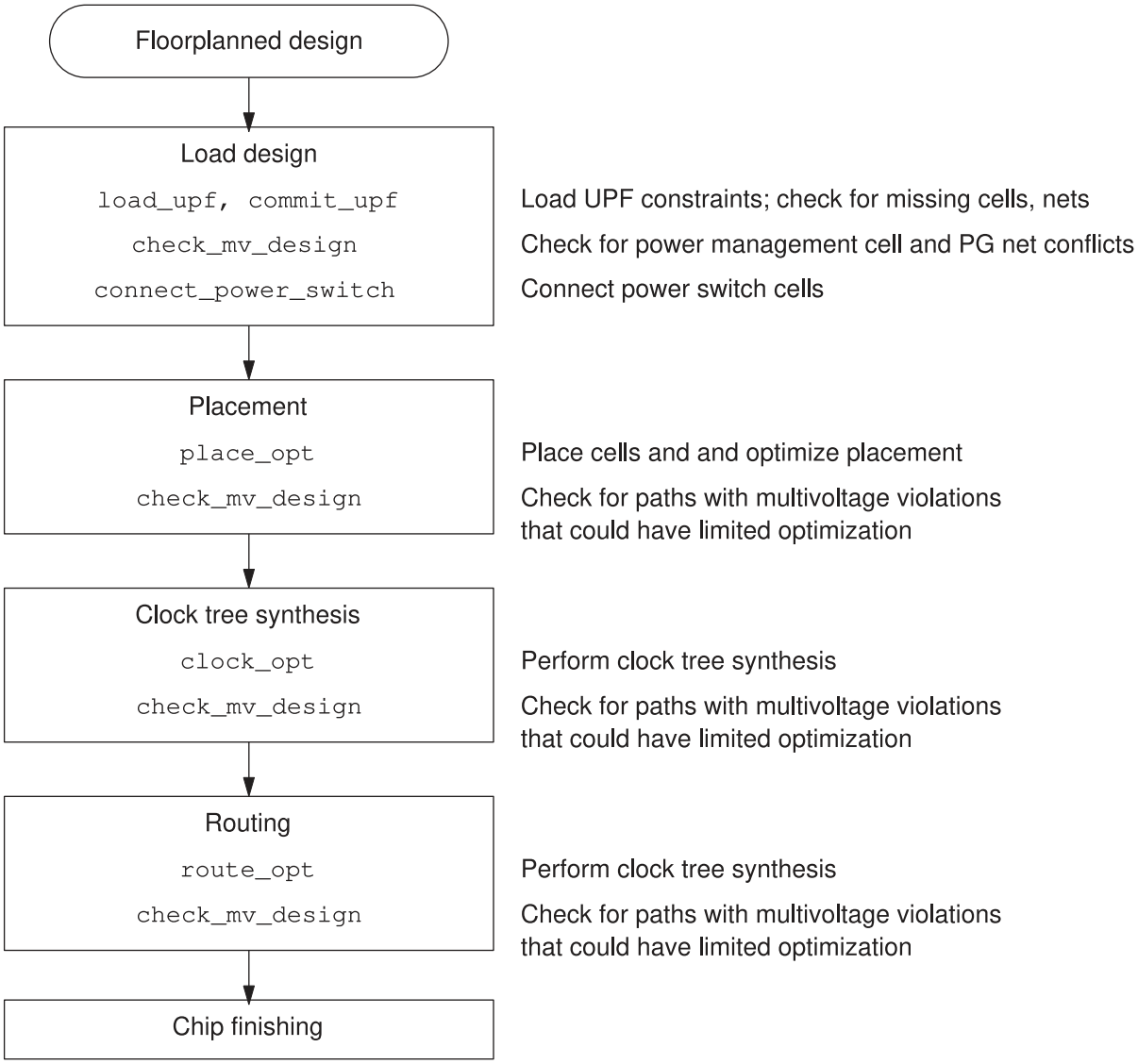
Physical Implementation Using IC Compiler II and Fusion Compiler

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 33 shows the general steps in the physical implementation flow and the low-power support features of the flow.

Feedback

Figure 33 IC Compiler II and Fusion Compiler Low-Power Physical Implementation Flow



For more information, see the following topics:

- Reference Library Setup
- Loading UPF Constraints
- Checking the Power Intent
- Existing Power Management Cells

- Inserting New Power Management Cells
- Specifying UPF Constraints for Physical-Only Cells
- Reporting and Debugging Power Intent
- Specifying Secondary PG Constraints
- Placement and Routing Optimization
- Standard Rail Power and Ground Connections
- Saving the Design and ASCII Export
- Hierarchical Flow in IC Compiler II or Fusion Compiler With UPF

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 `optimization` and `cts` so that the cells can be used for optimization and clock tree synthesis.

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 using the `report_attributes -class lib_cell` command.

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.

For information on modeling power management cells in the Liberty description language, see the following topics:

- *Library Requirements for Multivoltage Designs* in *Fusion Compiler™ Multivoltage User Guide* in Fusion Compiler™ Online Help
- *Advanced Low-Power Modeling* in *Library Compiler™ User Guide* in Library Compiler Online Help

Feedback

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.

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 contain only 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.

Checking the Power Intent

Checking the multivoltage-related infrastructure with the `check_mv_design` command is recommended after `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 the existing power management cells 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 with the power strategies defined in the UPF. To determine the power strategies for a power domain, use the `get_power_strategies` command.

You must 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 -generate_strategy level_shifter` command.

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 -filter` command.

To set the voltage for an operating corner, use the `set_voltage` command.

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.

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 core commands of the tool, `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 in to other tools that do not support physical-only cells, such as verification tools. 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.

Feedback

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 2 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_cells`	Main command to report detailed information on the power management cells in the 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.

Specifying Secondary PG Constraints

If you use dual-rail cells in a switched (or shutdown) power domain, the Fusion Compiler tool should place the cells near their secondary power straps. If the dual-rail cells are placed far away from the strap, the secondary PG routes might have too much IR drop or electromigration.

By default, the tool assumes that the secondary PG straps are available in all regions of a voltage area. However, to save routing resources, you might choose to have secondary

PG straps available in only a subset of the voltage area. In addition, the tool assumes that supplies are physically available as specified in the UPF.

Secondary PG constraints can help the tool to identify the location of straps that have limited physical availability, or no physical availability, in the voltage area. The tool honors these constraints throughout the flow. Dual-rail buffers affected by these constraints are placed in areas with secondary power straps. Dual-rail cells have priority over single-rail cells for placement in these areas.

You can specify secondary PG constraints manually. However, this effort might be difficult for a design with many supplies and voltage areas. Alternatively, the Fusion Compiler tool can evaluate the design and automatically derive basic secondary PG constraints, based on the strap locations.

For detailed information on manually and automatically creating the secondary PG constraints, checking the constraints, and resolving conflicts, see *Specifying Secondary PG Constraints* in the *Fusion Compiler™ Multivoltage User Guide* in the Fusion Compiler Online Help.

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.

For more information, see the following topics:

- Buffer Insertion
- Feedthrough Buffering
- Routing

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 Buffering

The Fusion Compiler tool optimizes always-on feedthrough nets that cross voltage areas due to differences in the logical and physical view of the design. However, the tool inserts regular buffers on always-on feedthrough nets, if doing so does not introduce electrical violations. Inserting regular buffers instead of always-on buffers improves QoR and reduces congestion.

By default, the tool inserts regular buffers on always-on feedthrough nets, instead of always-on buffers, if the primary supply of the feedthrough voltage area is more on than the sink supply and the primary supply is off when the source supply is off.

A physical feedthrough net is a feedthrough net that has different physical and logical topologies. The tool implements physical feedthrough buffering by using dual-rail buffers to buffer a feedthrough net across another domain without needing to punch ports.

In some cases, the tool cannot implement physical feedthrough buffering if the power domain in which the buffer resides is created at a scope below the net's logical hierarchy. In such a case, the tool implements a logical feedthrough buffer.

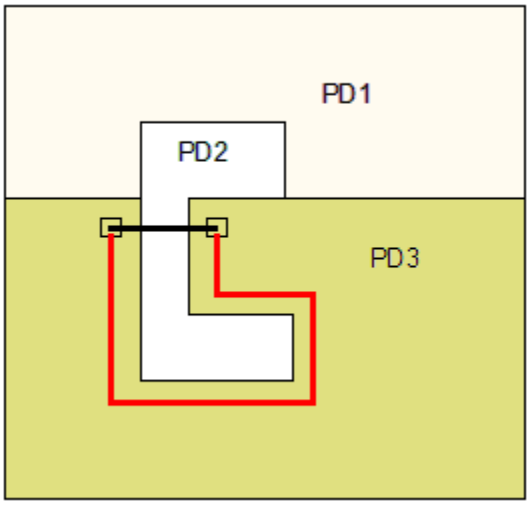
For detailed information on physical and logical feedthrough buffering and how to enable them, see *Feedthrough Buffering* in the *Fusion Compiler™ Multivoltage User Guide* in the Fusion Compiler Online Help.

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 34 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 following user guides:

- *IC Compiler™ II Design Planning User Guide* in the IC Compiler™ II Online Help
- *Fusion Compiler™ Design Planning User Guide* in the Fusion Compiler™ Online Help

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 the PrimeTime tool, you can write out the SPEF, SDC, and Verilog netlist data.

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.

Feedback

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 topics:

- Creating Block UPF From Full-Chip UPF
- Propagating Lower-Level Block UPF Into The Top Context
- Writing Full Chip-UPF From Lower-Level Blocks With UPF

For detailed information, see the following user guides:

- *IC Compiler™ II Design Planning User Guide* in the IC Compiler™ II Online Help
- *Fusion Compiler™ Design Planning User Guide* in the Fusion Compiler™ Online Help

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.

Feedback

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
```

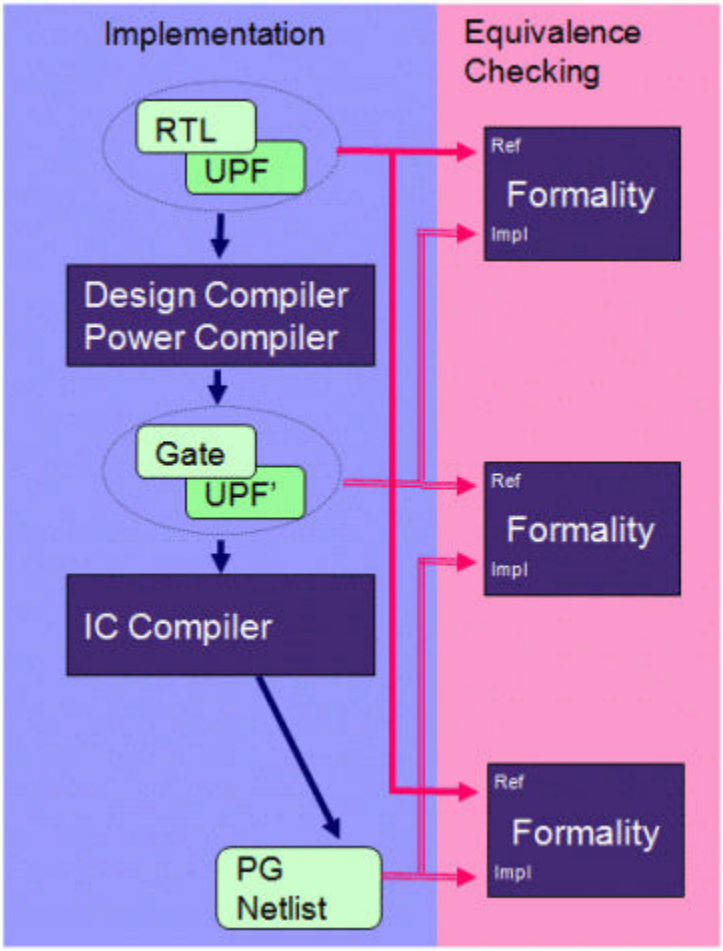
Formal Verification Using Formality

The Formality tool supports functional equivalence checking with low-power features such as clock gating, multiple-Vt, multivoltage supplies, power gating, and dynamic voltage and frequency scaling. The Formality tool recognizes low-power cells such as isolation cells, level shifters, always-on cells, retention registers, and power gates.

The Formality tool supports the verification of low-power design data with UPF as shown in Figure 35.

Feedback

Figure 35 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.

- 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.

Feedback

Formal verification requires certain preparation steps with respect to libraries and the treatment of level shifters and isolation cells. To run the Formality tool, 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 (the 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.

The Formality tool does not accept individual UPF commands entered interactively. The UPF description must be read in to the Formality tool 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, the Formality tool 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 report 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.
   Use 'report_libraries -defects errors | all' for more details.
**************************************************
```

To ensure accurate verification, you must correct these issues before you proceed. For the Formality tool to automatically fix some of these defects, in the setup mode set the `hdlin_library_autocorrect` variable to `true`, before reading the UPF file.

The Formality tool 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, the Formality tool verifies its functionality as usual. However, when a reference block is powered down, the compare points of the block are considered

Feedback

don’t care. The Formality tool 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.

The Formality tool 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, the Formality tool does not insert level shifters that perform only buffering. The Formality tool recognizes power state tables defined with the `create_pst` command.

For more information, see the *Formality® User Guide* in the Formality and Formality ECO Online Help.

For more information, see the following topics:

- Design Data Modification With UPF
- Retention Registers

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 `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 (“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.

Feedback

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.

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 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.

For more information, see the following topics:

- Voltage Scaling
- Setting Supply Voltages and Temperature

Feedback

- On-Chip Variation Analysis
- Reporting and Checking Multivoltage Designs

For detailed information on performing static timing analysis in the PrimeTime tool, see *The PrimeTime Static Timing Analysis Flow* in the *PrimeTime® User Guide* in the PrimeTime Suite Online Help.

For detailed information on power analysis using the PrimePower tool, see *Performing Power Analysis* in the *PrimePower and PrimePower RTL User Guide* in the PrimePower and PrimePower RTL Online Help.

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.

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 on voltage and temperature scaling, see *Cross-Library Voltage and Temperature Scaling* in the *PrimeTime® User Guide* in the PrimeTime Suite Online Help.

See also SolvNetPlus article 000024652, “PrimeTime Multivoltage Scaling Application Note.”

Feedback

Note:

You must be logged in to SolvNetPlus for the link to connect directly to the article. If you are prompted to log in to SolvNetPlus upon clicking the link to the article, log in, then click the link again to reach the article.

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 the PrimeTime tool, 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, if 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.

Feedback

On-chip variation analysis is recommended for timing signoff. 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.

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.

The `report_power_pin_info` command is useful for reporting the power pins of cells, including the voltage values.

To verify the signal-level consistency on the whole design, use the `check_timing` command.

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.

PrimeTime Signal Integrity (SI) supports static multivoltage crosstalk analysis, which deals with the voltage levels of the aggressor and victim nets. For more information, see *Signal Integrity Analysis* in the *PrimeTime® User Guide* in the PrimeTime Suite Online Help.

PrimePower Power Analysis

The PrimePower tool 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
```

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 `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.

For more information on multivoltage power analysis in the PrimePower tool, see *Multivoltage Power Analysis* in the *PrimePower User Guide* in the PrimePower Online Help.

PrimeRail Power Network Analysis

The PrimeRail tool extends the Synopsys signoff solution for power supply network integrity checking. The PrimeRail tool 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, the PrimeRail tool 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.

For more information, see the following topics:

- Power Network Analysis Flow
- Power-Up Inrush Current Analysis

See also the following information:

- PrimeRail User Guide
- SolvNetPlus article 000025442, “PrimeRail Reference Methodology”
- PrimeRail - Using Advanced Rail Analysis in the In-Design IC Compiler Implementation Flow

Note:

You must be logged in to SolvNetPlus for the link to connect directly to the article. If you are prompted to log in to SolvNetPlus upon clicking the link to the article, log in, then click the link again to reach the article.

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,

Feedback

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 in to the PrimeRail tool. While running rail analysis, the tool reads from these precharacterized models attached to the reference library.

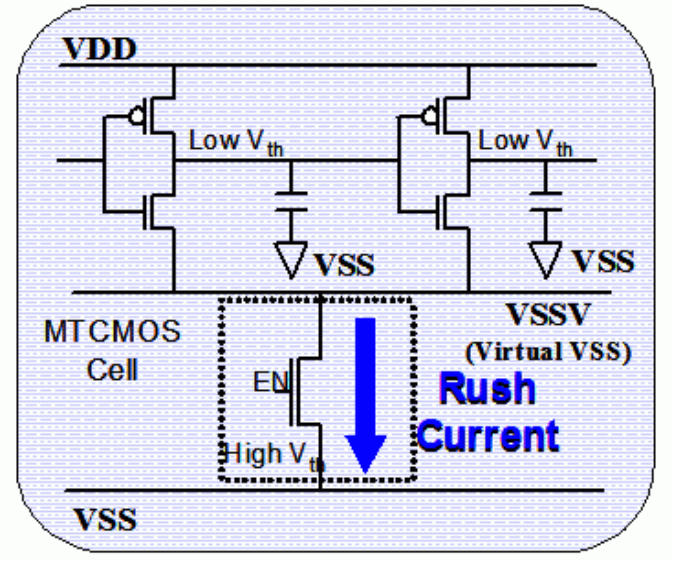
To generate rail analysis maps and current waveforms, you should run a gate-level power analysis to generate the necessary timing and power information. For this purpose, PrimePower is made accessible through PrimeRail. After this step, PrimeRail reads the binary report of power consumption analysis 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.

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 36. 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.

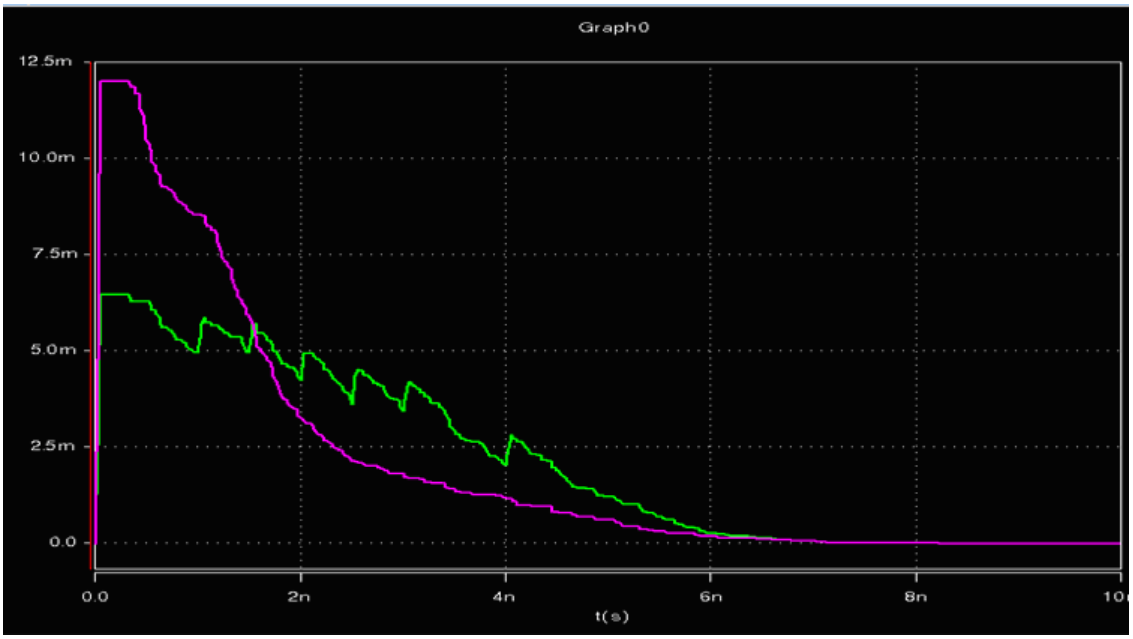
Figure 36 Rush Current Upon Power-Up



Feedback

PrimeRail provides the capability to analyze the rush current effects of MTCMOS circuits. Figure 37 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.

Figure 37 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 earlier. After extracting the 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.

7

Glossary

ABOR

Add Buffer On Route. Typically used as an Engineering Change Order operation where the default buffering results are insufficient. Refers to adding buffers with locations based on existing routing topology to avoid excessive disturbance to routing during postroute optimization. Can also be used to buffer special higher priority paths before the rest are done.

Active edge

The low-to-high or high-to-low transition of a clock signal that causes data to be latched into a flip-flop.

ADC

Automatic Density Control

ADCe

Automatic density control value enhanced

Aggressor net

A net that causes undesirable crosstalk effects on another net (called the victim net) due to parasitic cross-capacitance between the two nets.

AL

Advanced Legalizer

ANDR

Auto non-default-rules (NDR), non-default rules are created automatically by the tool during preroute or global-route based optimization, for the purposes of improving timing on a path or power on that net.

AO

The characteristic of a cell, circuit, or power domain that is on while a reference logic is shut down. For example, a logic cell used for isolation or retention is an always-

Feedback

on cell. The cell must have separate power supply pins for operation during power-down.

(always-on)

AOCV

Advanced on-chip variation, a technique to improve timing accuracy by applying derates to cells to model process variation. Less pessimistic than traditional on-chip variation, by using different derates based on cell type and path depth.

Arc

Short for timing arc. A representation of the timing relationship between two pins or ports in a circuit.

ASR

Automatic Scenario Reduction, a technique used during optimization to improve timer runtime by reducing the number of active scenarios.

Associated internal clocks

A set of one or more user-defined internal clocks in a clock network, each of which is the root of a skew subdomain within its parent clock network.

ATE

Automated Test Equipment, industrial equipment used to test semiconductor devices by applying input stimuli, observing the device response, and comparing it against the expected response.

ATPG

Automated Test Pattern Generation, the generation of scan data sequences used for scan testing, with the goal of achieving as much test coverage as possible using the smallest possible number of patterns. The test patterns contain nonfunctional data selected to detect faults on nets in the design.

Attribute

A string or value associated with an object that carries some information about that object. For example, the number_of_pins attribute attached to a cell indicates the number of pins in the cell. You can find out the values of attributes by using the `get_attribute` command.

BAP

Buffering-Aware Placement

Feedback

Behavioral view

The set of Verilog statements that describe the behavior of a design by using sequential statements. These statements are similar in expressive capability to those found in many other programming languages. See also the data flow view, sequential statement, and structural view definitions.

BIST

Built-in self-test, design-for-test logic where both the scan data generation and the scan data comparison logic are included in the design.

Bit-width

The width of a variable, signal, or expression in bits. For example, the bit-width of the constant 5 is 3 bits.

BSD

Boundary Scan Design, which refers to test logic that implements IEEE Std 1149.1.

BTSR

Boundary Track Supply Reduction

Burn-in mode

A mode that continuously runs autonomous self-test. The scan and capture activity stresses the tested logic and causes continuous power draw during self-test. Burn-in operation can be configured to stop or continue if self-test fails.

Bubble register

The internal part of a retention register that is always supplied with power and retains data during power-down; is constructed of higher-threshold transistors to minimize leakage current. Also called the shadow register.

(bubble register)

CBT

`create_buffer_tree`, this command builds buffer trees for the specified driver pins or nets.

cBuf

Post-route buffering engine

Feedback

CCD

Concurrent Clock and Data Optimization. Refers to useful skew techniques applied after clock tree synthesis to improve timing.

CDR

Core Data Register, an optional register in an IEEE 1500 implementation that can be loaded with a value after loading a corresponding instruction into the wrapper instruction register (WIR).

Cell

A component within an integrated circuit with known functional and timing characteristics, which can be used as an element in building a larger circuit.

CGPL

Conjugate Gradient Placer (coarse placer engine)

CL

Classic Legalizer

CLO

Concurrent Legalization and Optimization

Clock chain

A special scan chain segment associated with an on-chip clocking (OCC) controller whose scanned-in values control the pulse sequence of a controlled clock.

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 the transistors in the block sharing a single power switch.

(coarse-grain switching)

Codec

The combination of the decompressor and compressor in a compressed scan flow. A single design can have multiple codecs, but each codec consists of its own decompressor/compressor pair.

Feedback

Collection

A set of objects taken from the loaded design. For example, the set mylist `[get_clocks "CLK\*"]` command creates a collection of clocks and sets a variable called `mylist` to the collection. You can use commands such as `remove_clock $mylist` that operate on the collection. Additionally, you can also generate and operate on a collection within a single command, such as `remove_clock [get_clocks "CLK\*"]`.

Compressed scan

A scan methodology that uses more scan chains (called compressed scan chains) than scan-in/scan-out pairs. A decompressor decompresses the scan-in data to drive the greater number of scan chains. A compressor compresses the scan chain data to drive the lesser number of scan-outs. The combination of the decompressor and compressor wrapped around the scan chains is called the *codec*.

Compressed scan chains

In a compressed scan flow, the scan chains that are driven by the decompressor and drive the compressor. There are a greater number of compressed scan chains than scan-in/scan-out pairs.

Compressor

In a compressed scan flow, the part of a codec that compresses the scan chain data to drive the lesser number of scan-outs.

Core

A design block that is DFT-inserted and has CTL model information about the inserted DFT structures. Cores are used in hierarchical scan synthesis flows.

Core wrapping

See *wrapped core*.

CPU time

The total CPU processing time for a task. This is in contrast to elapsed time, which represents the actual amount of time that has elapsed during a task. When CPU time exceeds elapsed time, multithreaded commands or engines benefit from speedup that allows the work to be finished faster than if it were single threaded.

Feedback

Crowbar current

The current that flows from the power supply to the 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.

(crowbar current)

CTL model

Core Test Language model, describes the characteristics of a DFT-inserted design in STIL (IEEE Std 1450) format. This model includes information such as: DFT signals, scan chains, test clocks and clock control, and test modes.

Current Design

The active design (the design being worked on). Most commands are specific to the current design; they operate within the context of the current design. You specify the current design with the `current_design` command.

Current instance

The instance (hierarchical cell) in a design hierarchy on which instance-specific commands operate by default. Set the current instance with the `current_instance` command, which works like the `cd` command in Linux.

CUS

Compute Useful skew. Refer to the `compute_useful_skew` command, or any useful skew optimization done by the tool before clock trees are inserted. Contrast with CCD, which uses helpful skew operations on a clock tree that has already been constructed.

DCDP

Direct Congestion Driven Placement

DCLS

Dual Core Lock Step

Decompressor

In a compressed scan flow, the part of a codec that decompresses the scan-in data to drive the greater number of scan chains.

Design Constraints

A set of restrictions placed on the design to drive optimization goals like timing, power, and area. Timing constraints use SDC format. Many other types of constraints exist to

drive engines like clock tree synthesis (max skew), placement (bounds), optimization (low VT limits), and routing (non-default rules). Design constraints can be hard, meaning that they must be met or they are reported as a violation. They can also be soft, meaning that they are a best effort constraint to encourage the engine to do something if possible.

DFT

Design-for-Test, pertains to logic that helps the testability of a design.

DFTMAX scan compression

Scan compression implemented by the TestMAX DFT tool that uses combinational codecs to yield high scan compression ratios.

DFTMAX Ultra scan compression

Scan compression implemented by DFTMAX Ultra compression that uses streaming (sequential) codecs to yield very high scan compression ratios, even down to a single scan-in/scan-out pair.

DFT partition

A DFT specification that allows the sequential cells in a design to be separated into multiple independent partitions for the purpose of DFT insertion.

DMM

Design Mismatch Manager

DRC

Design Rule Checking, checking a design against a rule set that ensures good testability and reporting any violations of those rules.

DRO

Dynamic Rule Ordering for Legalization

DTDP

Direct Timing Driven Placement

Dynamic Power

Dynamic power is the energy consumed by CMOS circuits during switching (changing of logic states). It consists of two main components: switching power and internal power.

(dynamic power)

Feedback

Dynamic Voltage and Frequency Scaling

Dynamic voltage and frequency scaling is a speed-versus-power adjustment technique based on changing the supply voltage and operating frequency during operation to meet the current workload requirements. DVFS techniques are very effective in reducing power, since lowering the voltage has a squared effect on active power consumption. DVFS optimizes both the frequency and the voltage, it is one of the only techniques that is highly effective on both dynamic and static power.

DVFS (Dynamic Voltage and Frequency Scaling)

DVFS

Electronic Design Automation

Electronic design automation is the process of using advanced software tools to design integrated circuits.

EDA

EDC

Early Data Check

EDO

Enhanced Delay Optimization

eLPP

Enhanced Low Power Placement

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.

(enable level shifter)

Endpoint

The place in a design where a timing path ends. This is where data gets captured by a clock edge or where the data must be available at a specific time. Every endpoint must be either a register data input pin or an output port.

EPI

Enable Power Improvement

Feedback

ERI

Enable Runtime Improvements. Refers to the `enable_runtime_improvements` command, which enables code changes for runtime improvement that will be on-by-default in a future release.

EWI

Enable Wireopt Improvements. Refers to the `enable_wireopt_improvements` command, which enables wire optimization techniques like ANDR (auto-NDR) and layer promotion to improve timing and power, mostly during global-route based optimization in the `clock_opt` command.

Extent

The set of cells in a design that belong to a power domain.

(extent)

EXTEST mode

Outward-facing test mode used to test logic external to (outside) a core, independent of the logic inside the core. It uses the wrapper chain to drive scan-controllable values at the output wrapper cells and capture the values at the input wrapper cells.

Fanin

The cone of logic leads to a particular pin, port, net, or cell. Fanin means different things in different contexts. A fanin on a timing path is all of the logic on that path leading to a particular point. A fanin in a general timing context is all of the logic that can be traced in the timing graph upstream of that point. A fanin in the logical context is all of the logic that can be traced through the logical netlist connected to that point. A fanin in the context of a net is all of the pins or ports directly driving that net.

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.

(fine-grain switching)

FLL

Find Legal Location

Feedback

Footer switch

A power supply switch that connects the GND power supply to the GND supply pins of the switchable cells on the chip.

(footer switch)

FSM

Finite state machine, a logic construct that moves through states (like navigating elements in a flowchart diagram) to implement a certain logic behavior.

FTB

Feedthrough Buffering

FuSa

Functional Safety

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.

(gate leakage)

GDS

Graphic Design System

Golden UPF

In the golden UPF flow or mode, the same original *golden* UPF script file is used throughout the synthesis, physical implementation, and verification flow. Typically, this one same UPF file is referred to as the *golden* UPF. A more common flow is the UPF' or UPF prime flow, where the UPF file is modified across the flow with all the changes that the implementation front end and back end tools include.

(golden UPF)

GRO

Global Route Optimization. Refers to global-route based optimization. See also GRE.

GROPTO

Global Route based Optimization in HFS

Feedback

GUI

Graphical User Interface. A graphical interface for interacting with a software package. In the context of the tool, the GUI is launched with the `gui_start` command and enables many features for interactive editing and seeing the layout, schematic, or other views of the design.

HAB

Hold Area Budgeting

HAP

Hybrid (site-row) Aware Placement

hBuf

Hold Buffering Engine

HEA

High Effort Area

Header switch

A power supply switch that connects the supply VDD of the chip to the VDD pins of the switchable cells on the chip.

(header switch)

HEC

High Effort Congestion for `set_qor_strategy` (SQS) mega switch

HET

High Effort Timing

HFNS

High Fanout Net Synthesis

Hierarchical DFT insertion

Refers to a flow that performs DFT insertion in a lower level block (known as a *core*), then incorporates that block's scan structures into a higher level.

This term pertains to how DFT insertion is performed in the tool flow. Do not confuse this term with *hierarchical testing*, which pertains to how manufacturing test is run on the ATE.

Feedback

Hierarchical testing

Refers to the process of testing different hierarchy levels of the design independently. Wrapped cores are often used to enable hierarchical testing.

This term pertains to how manufacturing test is run on the ATE. Do not confuse this term with *hierarchical DFT insertion*, which pertains to how DFT insertion is performed.

Hold time

Hold time is the minimum amount of time that a signal on the data pin endpoint must remain stable after the active edge of the clock. The hold time creates a minimum delay requirement for paths leading to the data pin of the cell.

HPC

High Performance Core

HPWL

Half Perimeter Wire Length

HRO

hyper_route_opt. Also called as HROPT.

ICC2

IC Compiler II

ICG

Integrated Clock Gate. A clockgate with the latch mechanism built-in to the standard cell.

ICV

The IC Validator product from Synopsys

Ideal net

Ideal net is a network of cells that are assigned for ideal timing conditions. The timer does not compute arrival and transition times for these nets. Instead, ideal latency, transition, and capacitance constraints are applied. Ideal nets are typically used to model high-fanout nets before they are buffered, including very high-fanout data nets early in the synthesis flow (like scan nets), and clock trees before running clock tree synthesis, that is, latency, transition time, and capacitance are assigned a value of zero. The set_ideal_net command can constraint a net as ideal.

Feedback

Inout pin

An inout pin is a bidirectional pin of a cell that can function both as an input and an output.

Instance

An instance refers to a specific occurrence in a circuit of a reference (a library component or design) loaded in memory. Each instance has a unique name. A design can contain multiple instances where each instance points to the same reference but has a unique name to distinguish it from other instances. An instance is also known as a cell.

Internal chains

The scan chains inside a block (uncompressed or compressed), as opposed to other kinds of chains, such as boundary scan chains or core-wrapping chains.

Internal clock

A scan clock, defined on an internal pin in the design, that is the root driver of a *skew subdomain*. Internal clocks can be automatically determined by the tool at multi-input gate outputs or specified manually at associated internal clock pins within the clock network.

INTEST mode

Inward-facing test mode used to test logic internal to (inside) a core, independent of the logic outside the core. It uses the wrapper chain to drive scan-controllable values at the input wrapper cells and capture the resulting values at the output wrapper cells.

Internal power

Power consumed as a result of the short-circuit (crowbar) current through a PMOS-NMOS stack during a logic transition; is a type of dynamic power.

(internal power)

Insulated well

Usually refers to the N-well inside a cell that has some physical insulation that prevents physical abutment with its surrounding bias, allowing to be powered by a supply which behaves differently than the supply that powers the abutted N-wells in the physical area where the cell is going to be located.

(insulated well)

Feedback

Inward-facing test mode

See INTEST mode.

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.

(isolation cell)

LAP

Latency Aware Placement

LDP

Latency Driven Placement

Leading edge

The first edge of a clock waveform definition. It is a rising edge for a return-to-zero clock, and it is a falling edge for a return-to-one clock.

Leaf Cell

A fundamental unit of logic design. A leaf cell cannot be broken into smaller logic units. Examples are NAND gates and inverters. For example, standard cells and macro cells defined in a library are leaf cells. Contrast to a hierarchical cell.

LEE

Line End Extension

LEQ

Logic equivalency. Refers to checking done by any optimization engine that transforms netlist logic through operations like sizing and remapping. Ensures that the transformed cone of logic is logically equivalent to the same cone of logic before transformation.

Level shifter

A circuit or library cell that converts signals from one voltage level to another to make the signal compatible with the supply voltage of the power domain at the output.

(level shifter)

Feedback

LFSR

Linear feedback shift register, a shift register whose next data word is a linear XOR function of its current data word. The PRPG uses an LSFR to generate pseudorandom data.

LSF

Load Sharing Facility. A software package that handle scheduling and running jobs on a compute cluster. Many such software packages exist, LSF is often used as a generic term to refer to them.

MIB

Multiply Instantiated Block

MIM

Multiple Instantiated Modules

MISR

Multiple-input signature register, a recirculating shift register that XORs scan-captured data values into the loop. After capturing all values, the MISR contains a signature value for that test.

MLGR

Using Machine Learning technology to perform global route extraction

MSCTS

Multisource Clock Tree Synthesis, technique where a regular structure like an H-tree or clock mesh is used to drive the initial parts of the clock tree. The tool supports two main multisource CTS approaches such as structured multisource CTS (SMSCTS) and regular multisource CTS (RMSCTS)

Multiple-Threshold CMOS

Multiple-Threshold CMOS is 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.

MTCMOS (Multiple-Threshold CMOS)

Feedback

MTCMOS

Multiplexer

An N:1 multiplexer is a combinational circuit that connects any one input line, out of multiple N input lines, to the single output line based on its control input signal or selection lines. Usually, for n selection lines, there are $N = 2^n$ input lines. The nomenclature N:1 denotes it has N input lines and one output line.

MUX (Multiplexer)

MUX

Multivoltage

Multivoltage. Refers to a design technique where multiple power supplies are used as a power reduction technique. It can be different voltage or some power supplies that are fully switched off in certain conditions. Also, this refers to many techniques used in the tool to enable the multivoltage design. Multivoltage design constraints are written in a standard format called UPF.

MV

Multivoltage violation

Violations of multivoltage-related electrical rules of the design. This includes violations of rules related to power strategies: isolation, level shifter, retention, and repeater; violations of rules related with voltage thresholds in the design; violations derived from Power State Table (PST) analysis, among others.

(multivoltage violation)

MUXEO

Mux Extraction and Optimization

N-type Metal Oxide Semiconductor

Is 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.

NMOS (N-type Metal Oxide Semiconductor)

Feedback

NMOS

NDM

New Data Model

NDR

Non-Default Routing Rule

Netlist

A file in ASCII or binary format that describes a circuit schematic, the netlist contains a list of circuit elements and interconnections in a design. Netlist transfer is the most common way of moving design information from one design system or tool to another.

Non-Linear Delay Model

Is a highly accurate timing model derived from SPICE characterizations. Most cell libraries use look-up table models to specify the delay and timing checks for the various timing arcs of the cell. The table model is referred to as an NLDM and are used for calculating the delay, output slew, or other timing checks. The table gives the delay through the cell for various combinations of input transition time at the cell input pin and total output capacitance at the cell output. An NDLM delay model is represented in two-dimensional form, with two independent variables namely the input transition time and the output load capacitance and the entries in the table denote the delay.

NLDM (Non-Linear Delay Model)

NLDM

OBD

On By Default. A tool feature is on-by-default, so that there is no need to adjust an application option or other setting to enable it.

OCC controller

On-chip clocking controller, a DFT design structure that controls a free-running on-chip clocking source (such as a PLL output clock) in test mode.

Outward-facing test mode

See EXTEST.

PAC

Pin Access Checker

Feedback

Pad cell

A special cell at the chip boundaries that allows communication with other integrated circuits outside the chip, as opposed to an internal core cell, which makes up the core of an integrated circuit.

Parasitic Capacitance

Capacitance of a net due to the physical proximity between the net interconnections and adjacent nets or the substrate; or due to charge storage effects of p-n junctions in the net.

Parasitic Resistance

Resistance of an interconnection or net due to the finite conductivity of the interconnect material.

PBPAC

Pattern Based Pin Access Checker

PG pin

A power or ground pin of a cell, as specified in the library definition of the cell.

(PG pin)

Pin

A part of a cell that provides input and output connections. Pins can be bidirectional. The ports of a sub-design are pins within the parent design.

Pipelined scan data

A feature that inserts additional scan registers, called *pipeline registers*, at the scan-in and scan-out ports of the design to accommodate long wires between the scan chain input and the first flip-flop and between the last flip-flop and the scan chain output. Pipeline registers inserted at the scan-in and scan-out ports are called *head* and *tail* pipeline registers, respectively.

Pipelined scan enable

A feature that adds a pipeline register to the scan-enable signal. It is used for transition delay testing by making use of launch-on-extra-shift (LOES). This method of transition delay testing requires additional circuitry to manipulate the scan-enable signals.

Feedback

PLL

Phase-locked loop, an analog design block that creates a stable on-chip latency-adjusted clock from a free-running (and possibly less stable) input clock.

P-type Metal Oxide Semiconductor

Is 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.

PMOS (P-type Metal Oxide Semiconductor)

PMOS

Port

A signal declared in the interface list of an entity.

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 punching)

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.

(port state)

Post-DFT DRC

DRC checking run after DFT insertion, evaluates the implemented DFT functionality of the design for correct operation.

power

The rate of energy usage, usually expressed in watts (joules per second) and calculated as current (amps) multiplied by voltage drop (volts).

(power)

Feedback

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 domain)

Power domain boundary

Each design element, typically pins and ports, that is at the interface of two power domains. Power strategies are usually defined and applied at power domain boundaries.

(power domain boundary)

Power down

The process of shutting off the power to a part of the chip by turning off a power switch.

(power down)

Power intent

Equivalent to the functional intent of a design, the power intent is specified in a UPF file. The power intent specifies the power distribution architecture (power domains, supply rails, and shutdown control), the power states of a design (through voltage definitions and possible states of supplies according to the Power State Tables) and the usage of power management cells such as the isolation cells, level shifters, power switches, and retention registers.

(power intent)

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.

PST (Power State Table)

Feedback

PST

Power strategy

A rule specified by the user, that specifies where and how to apply isolation, level-shifting, state retention, and buffering in the implementation of the power intent.

(power strategy)

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 parts 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 the switching. The switch can optionally have multiple input control signals and one or more output acknowledge signals.

(power switch)

Pre-DFT DRC

DRC checking run before DFT insertion, evaluates the readiness of the design for DFT insertion. Certain types of pre-DFT DRC violations will result in scan cells being excluded from scan chains.

Primary supply

The supply net connections inferred for all instances in the power domain, unless overridden.

(primary supply)

Procedure (Tcl)

A procedure in the Tcl language is defined by the proc command, and is the same as what is called a function in most other programming languages.

Process, Voltage, and Temperature

To make chips to work in all the possible conditions after fabrication, they are simulated at different process, voltage, and temperature conditions. These conditions are called corners. All these three parameters directly affect the delay of the cell.

PVT (Process, Voltage, and Temperature)

Feedback

PVT

PRPG

Pseudo-random pattern generator, uses an LSFR and an XOR phase shifter to generate a stream of pseudorandom data values that have the appearance of random values, but are actually a function of a seed value.

RDE

Route Driven Estimation. An extraction engine to predict parasitics for a pre-routed design.

Receiver supply

For a receiver that is the gate of a transistor, the supply connected to the source or drain of that transistor. For a receiver that is the input to an inverter, the pair of supplies connected to the source or drain of the transistor pair comprising the inverter. For a receiver that is part of an active component, the primary supply of the power domain to which that receiver belongs or, in some cases, the secondary supply of the component if it has a secondary supply.

(receiver supply)

Reference

A library component or design that can be used as an element in building a larger circuit. The structure of the reference might be a simple logic gate or a more complex design (RAM core or CPU). A design can contain multiple occurrences of a reference; each occurrence is an instance. See also instance.

Register Transfer Level

Is a high-level description of a digital circuit expressed in terms of the data transfers between the registers and the logical operations performed on the data. The RTL description is written in a hardware description language such as Verilog or VHDL.

RTL (Register Transfer Level)

RTL

Related supply net

Supply net that is associated to a signal port or pin. In the case of a lib-pin, the association happens through the relations of signal pins and PG-pins of a lib-cell. In UPF, hierarchical pins can also have a related supply, which happens through the specification of UPF constraints (such as in power domain definitions,

Feedback

`set_port_attributes -driver_supply` command, or `set_port_attributes receiver_supply` command).

(related supply net)

Related supply set

Supply set that is associated to a signal port or pin. In the case of a lib-pin, the association happens through the relations of signal pins and PG-pins of a lib-cell. In UPF, hierarchical pins can also have a related supply, which happens through the specification of UPF constraints (such as in power domain definitions, `set_port_attributes -driver_supply` command, or `set_port_attributes receiver_supply` command).

(related supply set)

Return-to-one clock

A clock whose value is normally high, with an active-low pulse during the clock period.

Return-to-zero clock

A clock whose value is normally low, with an active-high pulse during the clock period.

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.

(retention register)

Reused supply net

A supply net that spans different domains and passes through a supply port.

(reused supply net)

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 with the `-scope` option of the command.

(root cell)

Feedback

RSPF

Reduced Standard Parasitic Format, a format used to store circuit parasitic information for back-annotation. This format is defined by Cadence Design Systems, Inc.

Scan cell

A sequential cell that has both functional and scan-shift modes of operation.

Scan compression

See *compressed scan*.

SCANDEF

A DEF file that uses a set of scan-specific constructs to describe how a design's scan chains can be reordered and repartitioned by a layout tool.

Scan group

A group of scan cells to be kept together in a scan chain.

SCO

Structural Congestion Optimization

Scope

The level of a design hierarchy where a power domain exists.

(scope)

Seed value

The initial value loaded into a PRPG. Different seed values result in different pseudorandom value sequences.

Shadow register

The internal part of a retention register that is always supplied with power and retains data during power-down. It is constructed using higher-threshold transistors to minimize leakage current. Is also called the bubble register.

(shadow register)

Shadow wrapper

A "wrapper" around an untestable block or macrocell that allows surrounding logic to be tested. Known values are forced at the outputs, and to improve coverage, the

Feedback

values at the inputs are captured. It can be easily inserted using automatic test point insertion.

Shared well

Usually refers to the N-well inside a cell that is physically abutted with the N-well of other cells in the surrounding region and shares electrical characteristics with them.

(shared well)

Shared codec I/O

A feature that allows the scan-in and scan-out connections of DFTMAX cores or codecs to be shared to reduce scan I/O requirements.

Shift register

A sequence of sequential cells in the design whose functional operation is to shift bits through the register like a scan chain. Shift registers only need their first (head) element scan-replaced to be stitched into a scan chain.

Skew subdomain

Part of a parent clock network that is considered to have different clock skew characteristics than the rest of the clock network. Lock-up latches are inserted whenever a scan chain crosses a skew subdomain boundary. A skew subdomain is driven by an *internal clock* pin.

SPEF

Standard Parasitic Exchange Format, a format used to store circuit parasitic information for back-annotation. This format is defined by Open Verilog International (OVI).

SPF

STIL Protocol File, a file written out by DFT Compiler to describe DFT aspects of the design, such as: test ports, test clocks, primary input constraints, scan chains, codecs, and test modes. It is used by TestMAX ATPG (or other ATPG tool) for DRC and ATPG.

SQS

This `set_qor_strategy` command is one of the mega switches in the tool for changing many engine configurations simultaneously. This command allows you to specify their primary design targets, whether that is timing, leakage power, total power, area, or congestion.

Feedback

Standard scan

A scan methodology in which there is a one-to-one relationship between each scan-in/scan-out pair and each scan chain.

STIL

Standard Test Interface Language, documented in IEEE Std 1450, which is a language used to describe the DFT capabilities of a design. It is a standard for simplifying the number of test vector formats that automated test equipment (ATE) vendors and computer-aided engineering (CAE) tool vendors must support.

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.

(static power)

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.

(sub-threshold leakage)

Supplemental UPF

In the golden UPF mode, a supplemental UPF file contains only the UPF changes made by the tool during the implementation session, such as tool-derived power intent and addition of power management cells. In the golden UPF flow, a supplemental UPF is required as the original golden UPF script file that is used throughout the synthesis, physical implementation and verification flow is used without incorporating any changes to it.

(supplemental UPF)

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 net)

Feedback

Supply port

A power supply connection point between two adjacent levels of a design hierarchy. A supply net that crosses from one level of the design hierarchy to the next passes through a supply port.

(supply port)

Supply set

An abstract representation of a bundle of supply nets that is specific to a power domain and that has a power and a ground function definition.

(supply set)

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.

(supply set handle)

Switch

A power switch; a device that turns on and turns off power for a supply net.

(switch)

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 during that period of time.

(switching activity)

Switching power

The power consumed as a result of charging and discharging the external capacitive load on the output of a cell.

(switching power)

Feedback

Switching Activity Interchange Format

A SAIF file is an ASCII file which captures the toggle rate of the signals in the design and is used for analyzing the power consumption in the design. SAIF is developed by Synopsys to facilitate the interchange of information between EDA simulation tools and Synopsys power tools. SAIF file can be generated from the Value Change Dump (VCD) file which is defined in the IEEE standard 1364-2001 to support the logging of signal and direction. VCD files are generated by the EDA simulation tools.

SAIF (Switching Activity Interchange Format)

SAIF

Tcl

Tool Command Language. The tool shell is based on the standard scripting language Tcl. Newer versions (U-2022.12-SP5 or later) of the tool also support Python shell to communicate with the tool.

TCM

Test control module, a DFT design structure that selects the current test mode. The input is a vector of test-mode selection signals that can have binary, one-hot, or user-defined encodings. The output is a set of decoded one-hot enable signals, one for each test mode encoding.

TDS

Timing Driven Structuring. Refers to logic restructuring steps in the optimization flow that target improving timing.

TEP

Targeted Endpoint Optimization. An option with `route_opt` to work only on a subset of the endpoints in the design. You can specify this option by using the `set_route_opt_target_endpoints` command.

THO

Transparent Hierarchy Optimization

Threshold Voltage

The gate-to-source voltage at which the transistor begins conducting between the source and drain.

(threshold voltage)

Feedback

Timing Path

A chain of logic through which data propagates. Data is launched by a clock edge at a startpoint, propagated through combinational logic elements, and captured at an endpoint by another clock edge. The startpoint of a timing path is an input port or clock pin of a sequential element. The endpoint of a timing path is an output port or a data pin of a sequential element.

TMCDR

Test-mode core data register, a special core data register (CDR) in an IEEE 1500 implementation that takes the place of traditional port-driven test-mode signals.

TNS

Total Negative Slack

tpLR

Total Power Logic Restructuring

TPO

Total Power Optimization

Trailing edge

The last edge of a clock waveform definition. It is a falling edge for a return-to-zero clock, and it is a rising edge for a return-to-one clock.

Transparent

The state of a level sensitive latch while the gate input is asserted. During this time, signals pass from input to output and the device operates like a buffer.

Type

In Verilog, the mechanism by which objects are restricted in the assigned values and the operations that can be applied to them.

UI

User Interface

Unified Power Format

Refers to a standard set of commands used to specify the low-power design intent for electronic systems. Is an alternative name of the IEEE 1801™ Standard for Design and Verification of Low-Power, Energy-Aware Electronic Systems.

Feedback

UPF (Unified Power Format)

UPF

Unified Power Format prime

In the traditional UPF flow used in synthesis tools, the UPF' file refers to the UPF file to which the `save_upf` command writes out a full set of UPF commands reflecting the changes made by the tool to the UPF power intent of the design. The file also includes the UPF commands run during the tool session.

UPF' (Unified Power Format prime)

UPF'

UPS

Unified Physical Synthesis. This refers to the optimization flow in the `compile_fusion` command. UPS brings advanced physically aware optimization earlier into the synthesis flow, and sophisticated logic restructuring techniques deeper into the place-and-route flow.

VDD

A name typically given to a power supply net or pin having a positive voltage.

(VDD)

VHDL

VHSIC Hardware Description Language. This language is used to model the behavior and structure of digital systems at multiple levels of abstraction, ranging from the system level down to the logic gates.

Victim net

A net that receives crosstalk effects due to parasitic cross-capacitance by another net (called the aggressor net).

Virtual Clock

A clock that exists on the chip but does not have a physical clock tree in this block. Typically used for I/O constraints, for example, a path might be launched by this virtual clock outside the block and come in one of its input ports. That input port's delay or clock latency might be constrained using this virtual clock.

Feedback

Voltage area

A physically contiguous area of the chip belonging to a single power domain and using the same power supply distribution network.

(voltage area)

VSS

A name typically given to a ground-voltage supply net or pin.

(VSS)

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.

(Vt)

Weighted capture groups

An OCC-based capture method that uses comparator logic to enable one capture group in each pattern, based on user-specified probabilities.

WIR

Wrapper instruction register, a required register in an IEEE 1500 implementation that controls what data register is selected for access.

WNS

Worst Negative Slack

Worker Process

In multithreaded jobs, one of a set of processes that performs distributed analysis work. The worker processes are controlled by the manager processes.

Wrapped core

A core that has a wrapper chain along the I/O boundary of the design. Wrapped cores are used to implement hierarchical testing capability, which allows different hierarchy levels of the design to be tested independently.

Feedback

Wrapper chain

A special scan chain in a core-wrapped design that consists of wrapper cells along the I/O boundary of the design. It can operate in inward-facing or outward-facing modes during testing to isolate the logic inside the core from logic outside the core.

ZRT

Zroute, which is the detailed router. Most of the router logfile messages use ZRT in the message tag.