Synopsys®
Low-Power Flow
User Guide

Version D-2009.12, December 2009

SYNOPSYS®

Copyright Notice and Proprietary Information

Right to Copy Documentation

The license agreement with Synopsys permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any. Licensee must assign sequential numbers to all copies. These copies shall contain the following legend on the cover page:

> "This document is duplicated with the permission of Synopsys, Inc., for the exclusive use of _____ and its employees. This is copy number _____."

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.

Registered Trademarks (®)

Synopsys, AMPS, Astro, Behavior Extracting Synthesis Technology, Cadabra, CATS, Certify, CHIPit, Design Compiler, DesignWare, Formality, HDL Analyst, HSIM, HSPICE, Identify, Leda, MAST, ModelTools, NanoSim, OpenVera, PathMill, Physical Compiler, PrimeTime, SCOPE, Simply Better Results, SiVL, SNUG, SolvNet, Syndicated, Synplicity, Synplify, Synplify Pro, Synthesis Constraints Optimization Environment, TetraMAX, the Synplicity logo, UMRBus, VCS, Vera, and YIELDirector are registered trademarks of Synopsys, Inc.

Trademarks (™)

AFGen, Apollo, Astro-Rail, Astro-Xtalk, Aurora, AvanWaves, BEST, Columbia, Columbia-CE, Confirma, Cosmos, CosmosLE, CosmosScope, CRITIC, CustomExplorer, CustomSim, DC Expert, DC Professional, DC Ultra, Design Analyzer, Design Vision, DesignerHDL, DesignPower, DFTMAX, Direct Silicon Access, Discovery, Eclypse, Encore, EPIC, Galaxy, Galaxy Custom Designer, HANEX, HAPS, HapsTrak, HDL Compiler, Hercules, Hierarchical Optimization Technology, High-performance ASIC Prototyping System, HSIM^{plus}, i-Virtual Stepper, IICE, in-Sync, iN-Tandem, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, Liberty, Libra-Passport, Library Compiler, Magellan, Mars, Mars-Rail, Mars-Xtalk, Milkyway, ModelSource, Module Compiler, MultiPoint, Physical Analyst, Planet, Planet-PL, Polaris, Power Compiler, Raphael, Saturn, Scirocco, Scirocco-i, Star-RCXT, Star-SimXT, StarRC, System Compiler, System Designer, Taurus, TotalRecall, TSUPREM-4, VCS Express, VCSi, VHDL Compiler, VirSim, and VMC are trademarks of Synopsys, Inc.

Service Marks (^{SM})

MAP-in, SVP Café, and TAP-in are service marks of Synopsys, Inc.

SystemC is a trademark of the Open SystemC Initiative and is used under license.
ARM and AMBA are registered trademarks of ARM Limited.
Saber is a registered trademark of SabreMark Limited Partnership and is used under license.
All other product or company names may be trademarks of their respective owners.

Contents

Preface

This preface includes the following sections:

What's New in This Release

The Eclypse™ Low Power Solution is a comprehensive portfolio of Synopsys tools that simplifies low-power design and verification by combining and automating a wide array of advanced techniques, methodologies, and standards. An important part of this flow is support for the IEEE 1801 Standard for Design and Verification of Low Power Integrated Circuits, also known as the Unified Power Format (UPF). Many Synopsys products support IEEE 1801 (UPF) infrastructure and commands, including Design Compiler, VCS, MVSIM, MVRC, Formality, IC Compiler, PrimeTime, and PrimeTime PX.

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

To see the release notes for a product,

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

   https://solvnet.synopsys.com/DownloadCenter

   If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.
2. Select the Synopsys product, and then select a release in the list that appears.

About This Guide

This user guide describes the flow of the Eclypse Low Power Solution, including synthesis, implementation, and verification. It serves as a guide to the flow and IEEE 1801 (UPF) usage in various Synopsys tools. Detailed information on the standard can be found in the IEEE 1801 specification itself, available from IEEE Xplore at http://ieeexplore.ieee.org; or from the UPF Standard 1.0, available from Accellera at http://www.accellera.org. Detailed information about Synopsys product support for the low-power flow is available in the product manuals for individual Synopsys tools.

Audience

This manual is intended for engineers who design integrated circuits using one or more of the tools of the Eclypse Low Power Solution, Galaxy Design Platform, and Discovery Verification Platform, and who employ reduced-power design techniques such as gated clocks, multivoltage power domains, power-down domains, and multiple-threshold cells.

Related Publications

For additional information about Synopsys products, see Documentation on the Web, which is available through SolvNet at the following address:

https://solvnet.synopsys.com/DocsOnWeb

Conventions

The following conventions are used in Synopsys documentation.

Table 1

Convention	Description
`Courier`	Indicates command syntax.
`Courier italic`	Indicates a user-defined value in Synopsys syntax, such as *`object_name`*. (A user-defined value that is not Synopsys syntax, such as a user-defined value in a Verilog or VHDL statement, is indicated by regular text font italic.)
`Courier bold`	Indicates user input—text you type verbatim—in Synopsys syntax and examples. (User input that is not Synopsys syntax, such as a user name or password you enter in a GUI, is indicated by regular text font bold.)
[ ]	Denotes optional parameters, such as *`pin1`* [*`pin2 ... pinN`*]
|	Indicates a choice among alternatives, such as `low | medium | high` (This example indicates that you can enter one of three possible values for an option: low, medium, or high.)
_	Connects terms that are read as a single term by the system, such as `set_annotated_delay`
Control-c	Indicates a keyboard combination, such as holding down the Control key and pressing c.
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy.

Customer Support

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

Accessing SolvNet

SolvNet includes an electronic knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. SolvNet also gives you access to a wide range of Synopsys online services including software downloads, documentation on the Web, and “Enter a Call to the Support Center.”

To access SolvNet, go to the SolvNet Web page at the following address:

https://solvnet.synopsys.com

If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.

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

Contacting the Synopsys Technical Support Center

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

- Open a call to your local support center from the Web by going to https://solvnet.synopsys.com (Synopsys user name and password required), and then clicking “Enter a Call to the Support Center.”
- Send an e-mail message to your local support center.
  - E-mail support_center@synopsys.com from within North America.
  - Find other local support center e-mail addresses at http://www.synopsys.com/Support/GlobalSupportCenters/Pages
- Telephone your local support center.
  - Call (800) 245-8005 from within the continental United States.
  - Call (650) 584-4200 from Canada.
  - Find other local support center telephone numbers at http://www.synopsys.com/Support/GlobalSupportCenters/Pages

1

Low-Power Design Strategies

Power consumption is becoming an increasingly important aspect of circuit design. There are several different approaches that can be used to reduce power. This chapter introduces power principles and power reduction strategies in the following sections:

Increasing Challenges of Power

In earlier generations of IC design technologies, the main parameters of concern were timing and area. EDA tools were designed to maximize the speed while minimizing area. Power consumption was a lesser concern. CMOS was considered a low-power technology, with fairly low power consumption at the relatively low clock frequencies used at the time, and with negligible leakage current.

In recent years, however, device densities and clock frequencies have increased dramatically in CMOS devices, thereby increasing the power consumption dramatically. At the same time, supply voltages and transistor threshold voltages have been lowered, causing leakage current to become a significant problem. As a result, power consumption levels have reached their acceptable limits, and power has become as important as timing or area.

High power consumption can result in excessively high temperatures during operation. This means that expensive ceramic chip packaging must be used instead of plastic, and complex and expensive heat sinks and cooling systems are often required for product operation. Laptop computers and hand-held electronic devices can become uncomfortably hot to the touch. Higher operating temperatures also reduce reliability because of electromigration and other heat-related failure mechanisms.

High power consumption also reduces battery life in portable devices such as laptop computers, cell phones, and personal electronics. As more features are added to a product, power consumption increases and the battery life is reduced, requiring a larger, heavier battery or shorter life between charges. Battery technology has lagged behind the increased demands for power.

Another aspect of power consumption is the sheer cost of electrical energy used to power millions of computers, servers, and other electronic devices used on a large scale, both to run the devices themselves and to cool the machines and buildings in which they are used. Even a small reduction in power consumption of a microprocessor or other device used on a large scale can result in large aggregate cost savings to users and can provide significant benefits to the environment as well.

Dynamic and Static Power

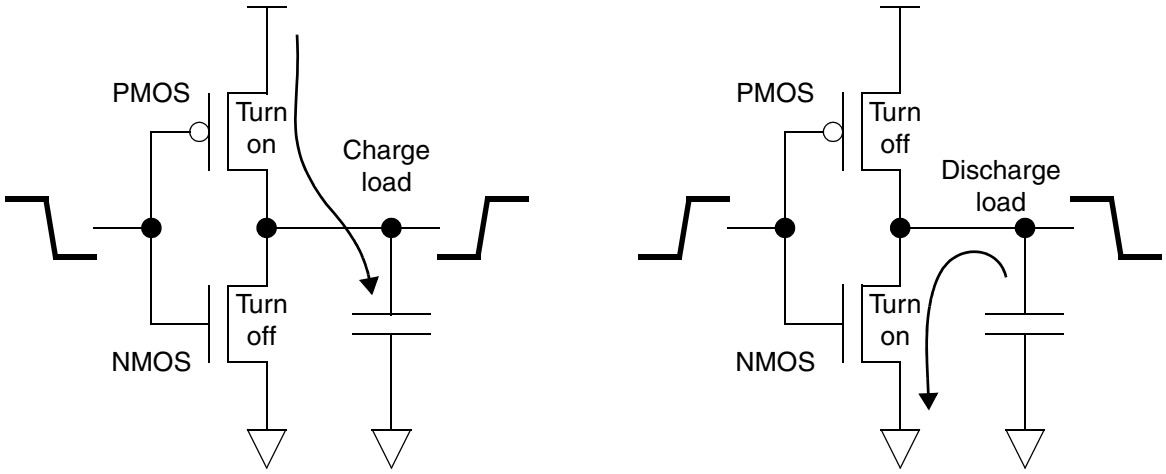
Designers must consider two types of power consumption, dynamic and static. Dynamic power is consumed during the switching of transistors, so it depends on the clock frequency and switching activity. Static power is the transistor leakage current that flows whenever power is applied to the device, so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

Dynamic Power

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

The cause of switching power is illustrated in Figure 1-1. A transition from 0 to 1 on the output of the inverter charges the capacitive load of the output net through the PMOS transistor. A transition from 1 to 0 discharges the same capacitive load through the NMOS transistor.

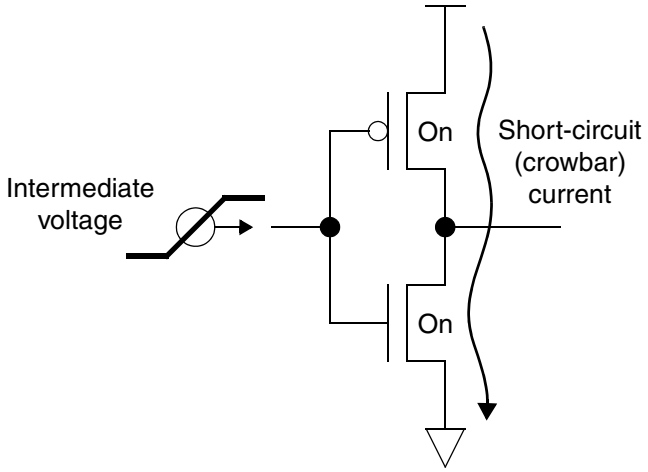
Figure 1-1 Switching Power



The amount of energy dissipated in each transition depends on the supply voltage and the capacitive load of the net. Also, because the current flows only during logic transitions on the net, the long-term dynamic power consumption depends on the clock frequency (possible transitions per second) and the switching activity (presence or absence of transitions actually occurring on the net in successive clock cycles).

Internal power is consumed during the short period of time when the input signal is at an intermediate voltage level, during which both the PMOS and NMOS transistors can be conducting. This condition results in a nearly short-circuit conductive path from VSS to ground, as illustrated in Figure 1-2. A relatively large current, called the crowbar current, flows through the transistors for a brief period of time. Lower threshold voltages and slower transitions result in more internal power consumption.

Figure 1-2 Internal Power

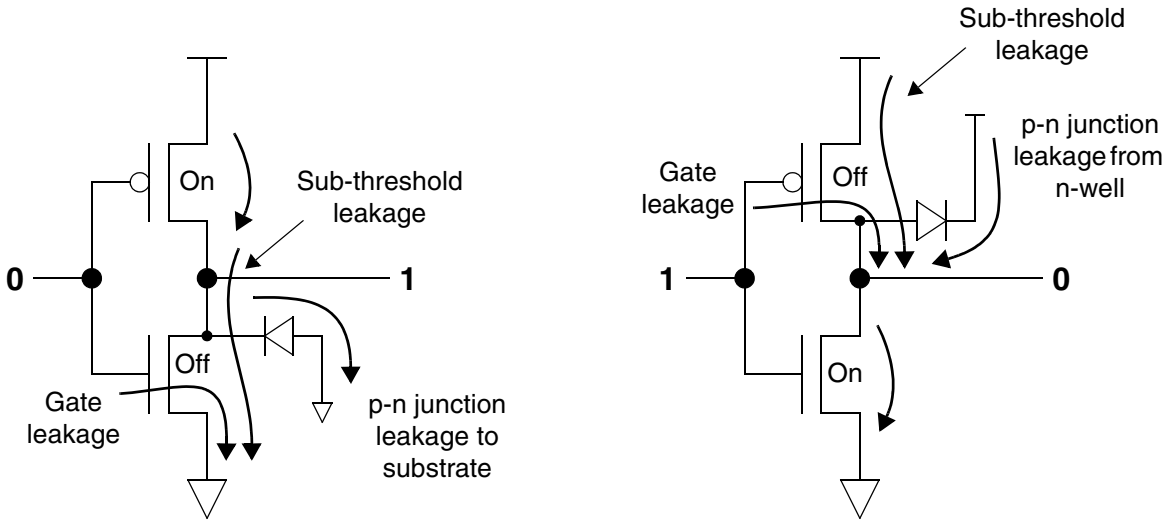


Static (Leakage) Power

Leakage current was negligible in earlier CMOS technologies. However, with shrinking device geometries and reduced threshold voltages, leakage power is becoming increasingly significant, sometimes approaching the levels of dynamic power dissipation.

The main causes of leakage power are reverse-bias p-n junction diode leakage, sub-threshold leakage, and gate leakage. These leakage paths in a CMOS inverter are shown in Figure 1-3.

Figure 1-3 Static Leakage Currents



Leakage at reverse-biased p-n junctions (diode leakage) has always existed in CMOS circuits. This is the leakage 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.

Sub-threshold leakage is the small source-to-drain current that flows even when the transistor is held in the “off” state. In older technologies, this current was negligible. However, with lower power supply voltages and lower threshold voltages, “off” gate voltages are getting close to “on” threshold voltages. Sub-threshold leakage current increases exponentially as the gate voltage approaches the threshold voltage.

Gate leakage is the result of using an extremely thin insulating layer between the gate conductor and the MOS transistor channel. Gate oxides are becoming so thin that only a dozen or fewer layers of insulating atoms separate the gate from the source and drain. Under these conditions, quantum-effect tunneling of electrons through the gate oxide can occur, resulting in significant leakage from the gate to the source or drain.

Leakage currents occur whenever power is applied to the transistor, irrespective of the clock speed or switching activity. Leakage cannot be reduced by slowing or stopping the clock. However, it can be reduced or eliminated by lowering the supply voltage or by switching off the power to the transistors entirely.

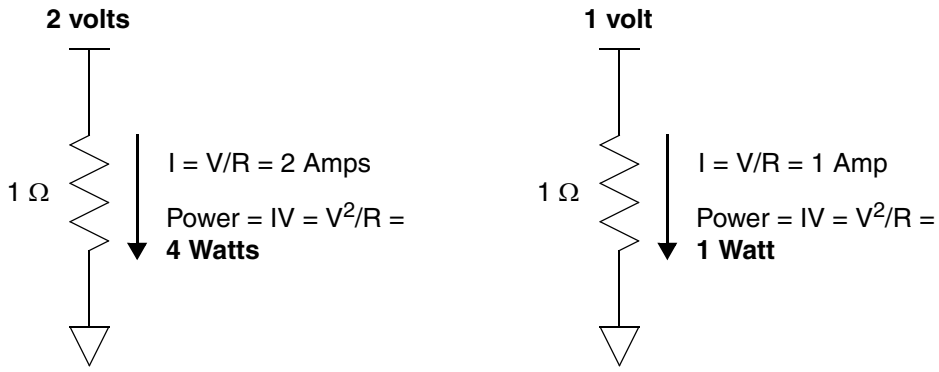
Power Reduction Methods

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 in the past due to the difficultly of implementing them. As power becomes increasingly important in advanced technologies, more methods are being exploited to achieve the design requirements.

Supply Voltage Reduction

The most basic way to reduce power is to reduce the supply voltage. Power usage is proportional to the square of the supply voltage, as the simple example in Figure 1-4 demonstrates. A 50 percent reduction in the supply voltage results in a 50 percent reduction in current and a 75 percent reduction in power. A similar degree of power reduction can be achieved in CMOS circuits for both dynamic and static power.

Figure 1-4 Power Versus Supply Voltage Example

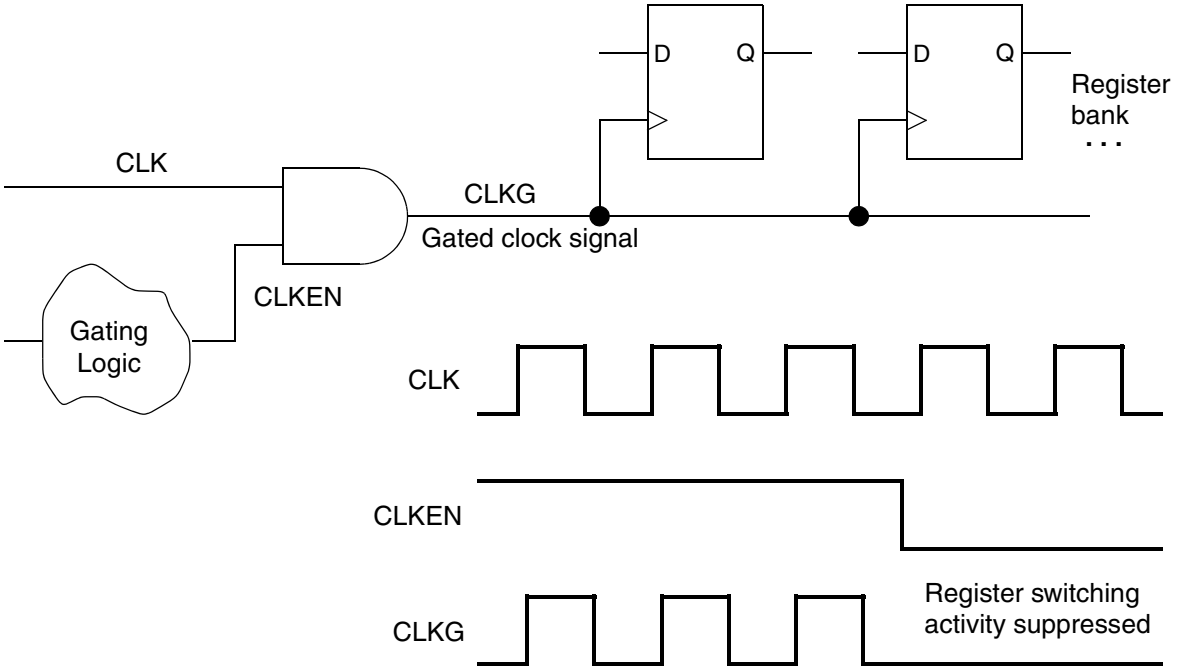


Successive generations of CMOS technologies have used lower and lower supply voltages to reduce power. Starting at 5 volts for compatibility with bipolar TTL circuits in the 1980s, supply voltages have fallen to about 1 volt for advanced technologies. Each lowering of the supply voltage reduces per-gate 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. The lower voltage swing makes it more difficult to interface the chip with external devices that require larger voltage swings.

Clock Gating

Clock gating is a dynamic power reduction method in which the clock signals are stopped for selected register banks during times when the stored logic values are not changing. One possible implementation of clock gating is shown in Figure 1-5.

Figure 1-5 Clock Gating Example



Clock gating is particularly useful for registers that need to maintain the same logic values over many clock cycles. Shutting off the clocks eliminates unnecessary switching activity that would otherwise occur to reload the registers on each clock cycle. The main challenges of clock gating are finding the best places to use it and creating the logic to shut off and turn on the clock at the proper times.

Clock gating is a well-established power-saving technique that has been used for years. Synthesis tools such as Power Compiler can detect low-throughput datapaths where clock gating can be used with the greatest benefit, and can automatically insert clock-gating cells in the clock paths at the appropriate locations. Clock gating is relatively simple to implement because it only requires a change in the netlist. No additional power supplies or power infrastructure changes are required.

Multiple-Vt Library Cells

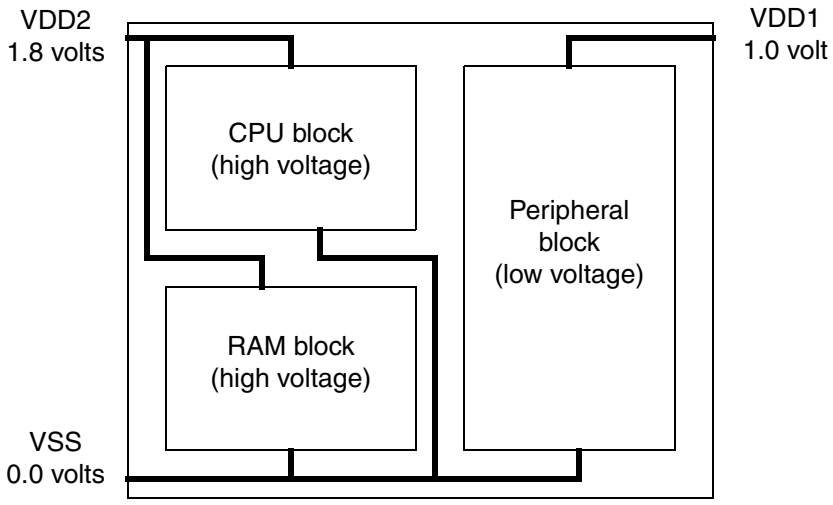
Some CMOS technologies support the fabrication of transistors with different threshold voltages (Vt values). In that case, the cell library can offer two or more different cells to implement each logic function, each using a different transistor threshold voltage. For example, the library can offer two inverter cells: one using low-Vt transistors and another using high-Vt transistors.

A low-Vt cell has higher speed, but higher sub-threshold leakage current. A high-Vt cell has low leakage current, but less speed. The synthesis tool can choose the appropriate type of cell to use based on the tradeoff between speed and power. For example, it can use 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. As mentioned earlier, a lower supply voltage reduces power consumption but also reduces speed. To get maximum speed and lower power at the same time, the CPU and RAM can operate with a higher supply voltage while the peripheral block operates with a lower voltage, as shown in Figure 1-6.

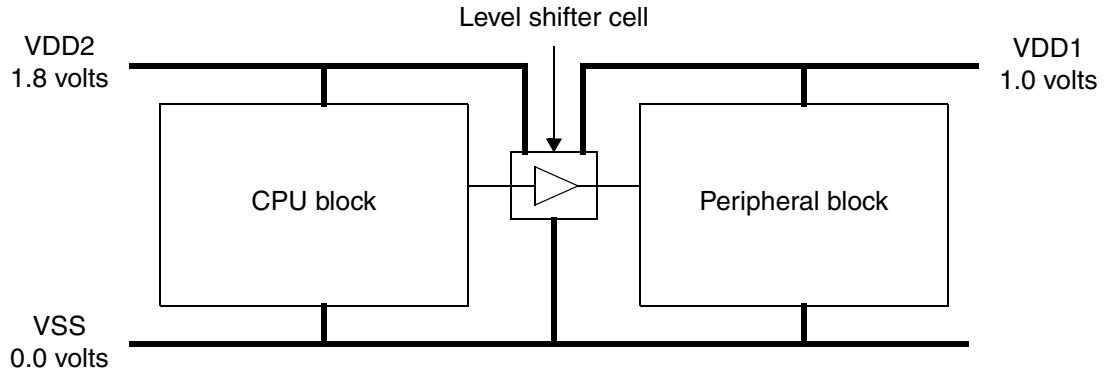
Figure 1-6 Multivoltage Chip Design



Providing two or more 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 1-7, 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.

Figure 1-7 Level Shifter



Power Switching

Power switching is a power-saving technique in which portions of the chip are shut down completely during periods of inactivity. For example, in a cell phone chip, the block that performs voice processing can be shut down when the phone is in standby mode. When the user places a call or receives an outside call, the voice processing block must “wake up” from its powered-down state.

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, including the need for a power controller, a power-switching network, isolation cells, and retention registers.

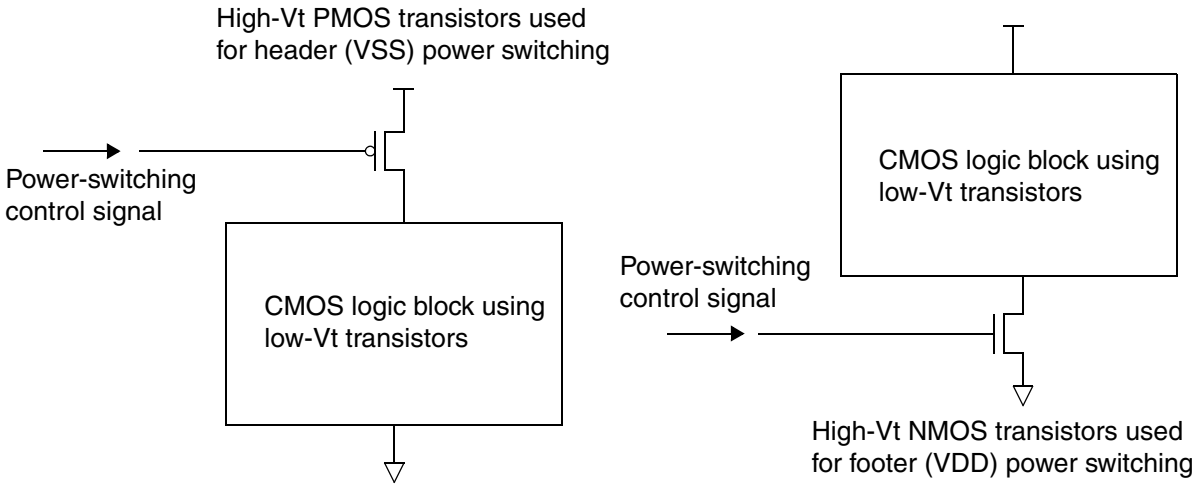
A power controller is a logic block that determines when to power down and power up a specific block. There is a certain amount of time and power cost for powering down and powering up a block, so the controller should determine the appropriate power-down times with a high degree of certainty.

A block that can be powered down must receive its power through a power-switching network, consisting of a larger number of 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. When switched off, the power supply is effectively disconnected from the logic gates in the block.

High-Vt transistors from a Multiple-Threshold CMOS (MTCMOS) technology are used for the power switches because they minimize leakage and their switching 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 1-8. The number, drive strength, and placement of switches should be chosen to give in an acceptable voltage drop during peak power usage in the block.

Figure 1-8 Power-Switching Network Transistors

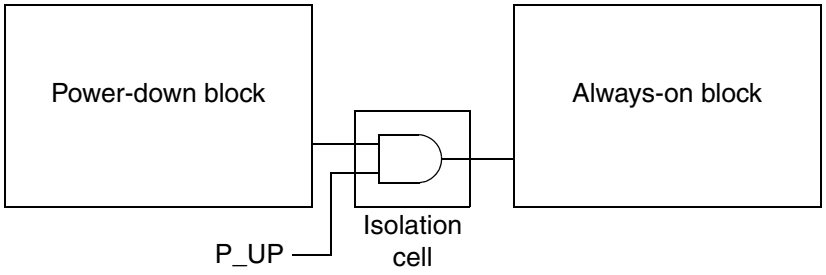


The switching strategy shown in Figure 1-8 is called a coarse-grain strategy because the power switching is applied to the whole block. Multiple transistors in parallel drive a common supply net for the block. In a fine-grain strategy, each library cell has its own power switch, allowing fine-grain control over which cells are powered down. The fine-grain approach has better potential for power savings, but requires significantly more area.

Any use of power switching requires isolation cells where signals leave a powered-down block and enter a block that is always on (or currently 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.

One simple implementation of an isolation cell is shown in Figure 1-9. When the block on the left is powered up, the signal P_UP is high and the output signal passes through the isolation cell unchanged (except for a gate delay). When the block on the left is powered down, P_UP is low, holding the signal constant at logic 0 going into the always-on block. Other types of isolation cells can hold a logic 1 rather than 0, or can hold the signal value latched at the time of the power-down event. Isolation cells must themselves have power during block power-down periods.

Figure 1-9 Isolation Cell



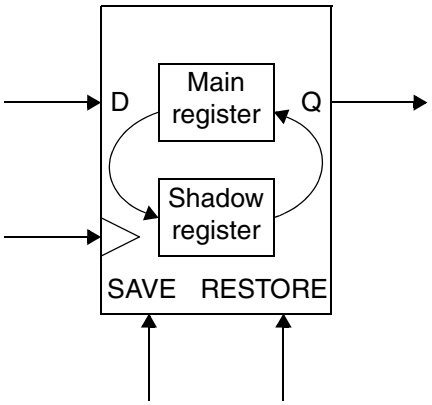
The power switching can be combined with multivoltage operation. Different blocks can be designed to operate at different voltages and also to be separately powered down when they are not needed. In that case, the interface cells between different blocks must perform both level shifting and isolation 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.

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 prior to 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. A retention register can retain data during power-down by saving the data into a shadow register (also known as the bubble register) prior to power-down. Upon power-up, it restores the data from the shadow register to the main register. The shadow register has an always-on power supply, but it is constructed with high-Vt transistors to minimize leakage during the power-down period. The main register is built with fast but leaky low-Vt transistors.

One type of retention register implementation is shown in Figure 1-10. The SAVE signal saves the register data into the shadow register prior to 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 1-10 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 in a laptop computer might operate at a lower voltage and lower clock frequency during simple spreadsheet computations, thereby saving power; and then at a higher voltage and higher clock frequency during 3-D image rendering when the highest performance is needed. The changing of supply voltage and operating frequency during operation 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 especially challenging because of the ranges and combinations of voltage levels and operating frequencies that must be analyzed and accommodated.

Dynamic voltage scaling can be combined with power switching technology so that each block in the design can operate at multiple voltage levels for different performance requirements, or shut off completely when not needed at all.

2

Library Requirements

In order to use power-saving strategies such as clock gating, multivoltage, multiple-Vt library cells, or power switching, the technology 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. Technology 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 tools to optimize the design for power, to properly connect the cells in layout, and to analyze the design behavior where multiple supply voltages are being used.

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

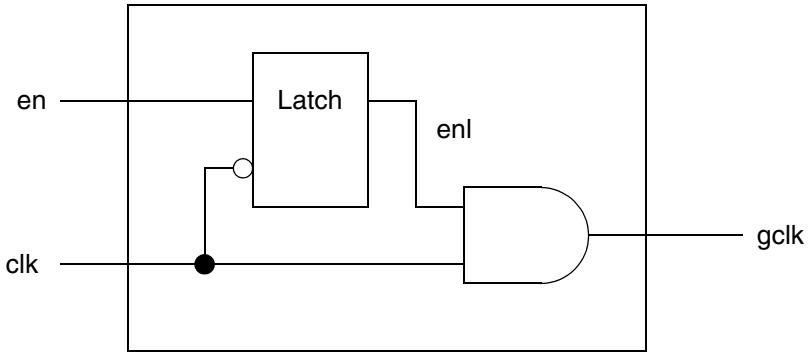
For an older library that does not have PG pins, you can quickly add PG pins with the `add_pg_pin_to_lib` or `add_pg_pin_to_db` command in Design Compiler or IC Compiler, thereby making the library compatible with UPF power specification. For details, see the man pages for the commands.

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 2-1 shows one possible implementation of an integrated clock-gating cell.

Figure 2-1 Integrated Clock-Gating Cell Example



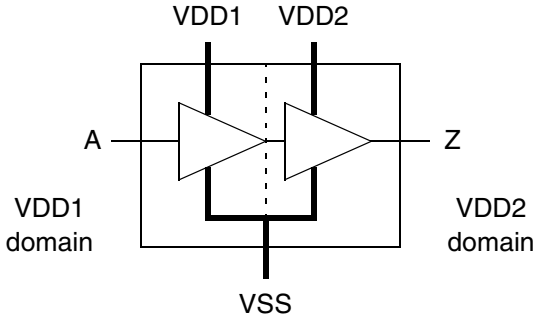
A clock-gating cell can incorporate any kind of logic such as multiple enable inputs, test clock input, global scan input, asynchronous reset latch, active-low enabling logic, or inverted gated clock output. Power Compiler is free to optimize the enabling logic surrounding a clock-gating cell by absorbing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

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

Level Shifters

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 2-2.

Figure 2-2 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.

Synthesis and implementation tools such as Power Compiler and IC Compiler 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);
  }
```

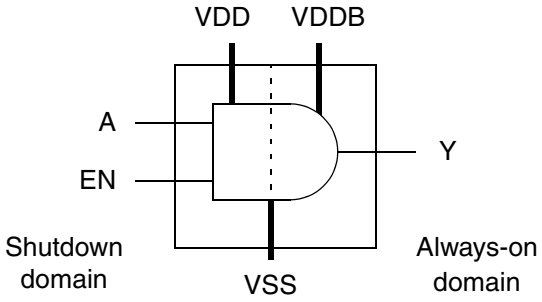
```
 pin(Z) {
   direction : output;
   related_power_pin : VDD2;
   related_ground_pin : VSS;
   function : "A";
   power_down_function : "!VDD1 + !VDD2 + VSS";
   output_voltage_range (1.1 , 1.3);
   ...
...
}
```

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

Isolation Cells

In a design with power switching, an isolation cell is required where each logic signal crosses from a power domain that can be powered down to a domain that is not powered down. The cell operates as a buffer when the input and output sides of the cell are both powered up, but provides a constant output signal during times that the input side is powered down. An enable input controls the operating mode of the cell. See Figure 2-3.

Figure 2-3 Isolation Cell



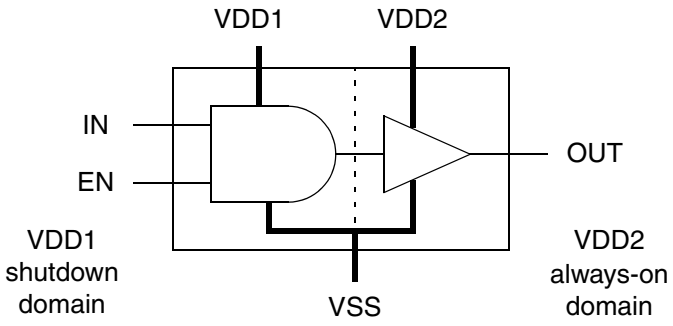
The following example shows the form of the Liberty syntax for a typical isolation cell.

```
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 2-4.

Figure 2-4 Enable Level Shifter



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

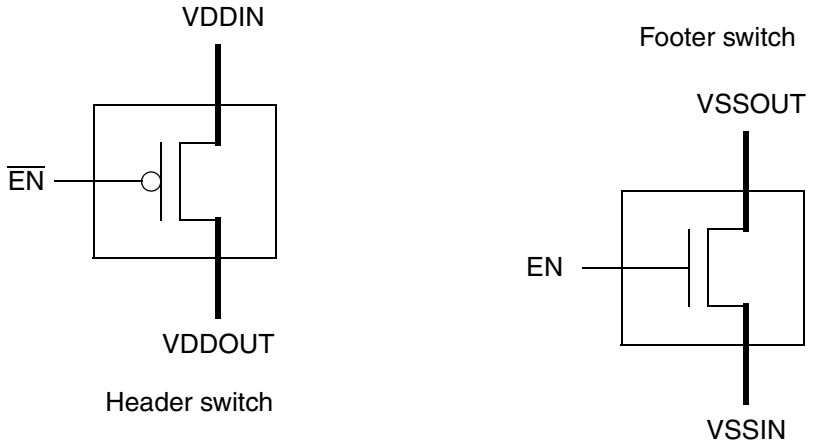
```
cell(Enable_Level_Shifter) {
  is_level_shifter : true;
  level_shifter_type : LH ;
  input_voltage_range(0.7,1.4);
  output_voltage_range(0.7,1.4);

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;
}
...
```

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

Power-Switch Cells

In a design with power switching, either header or footer type power switch cells are required to supply power for cells that can be powered down. A header type power switch connects the power rail to the power supply pins of the cells in the power-down domain. A footer type power switch connects the ground rail to the ground supply pins of the cells in the power-down domain. An input logic signal to the power switch controls the connection or disconnection state of the switch. See Figure 2-5.

Figure 2-5 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.

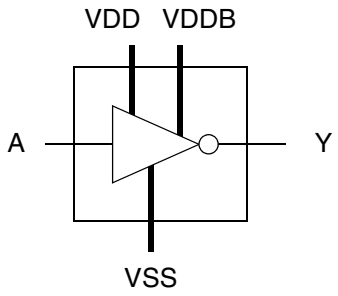
```
cell ( Simple_CG_Switch ) {
  ...
  switch_cell_type: coarse_grain;

  pg_pin ( VDDG ) {
    pg_type :primary_power;
    direction : input;
    voltage_name : VDD;
  }
 ...
   pg_pin(VDD) {
     voltage_name : VDD;
     pg_type:internal_power;
     direction : inout;
     switch_function : "SLEEP";
     pg_function : "VDDG";
  }
 ...
 pin ( SLEEP ) {
    switch_pin : true;
    capacitance: 0.034;
  }
 ...
```

For more information on creating power-switch cells, see the chapter called “Advanced Low Power Modeling” in the *Library Compiler Modeling Timing, Signal Integrity, and Power in Technology Libraries User Guide*. For information on using these cells in physical planning, see the *IC Compiler Design Planning User Guide*.

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 2-6.

Figure 2-6 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";
}
...
```

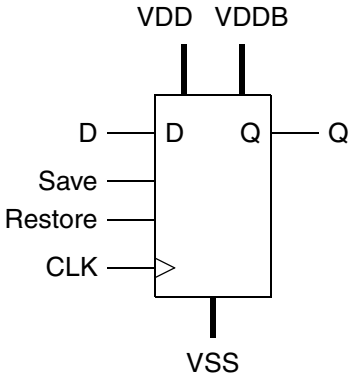
For more information on creating and using always-on logic cells, see the chapter called “Advanced Low Power Modeling” in the *Library Compiler Modeling Timing, Signal Integrity, and Power in Technology Libraries User Guide*; and the *Power Compiler User Guide*.

Retention Register Cells

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

The library description of a retention register specifies the power pins and the input signals that control the saving and restoring of data. It also specifies which power pins are normal and can be powered down and which ones are the always-on pins used to maintain the data during power-down. See Figure 2-7.

Figure 2-7 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" );
    }
  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 the chapter called “Retention Register Implementation” in the *Power Compiler User Guide*.

3

Power Intent Specification

The IEEE 1801 Standard for Design and Verification of Low Power Integrated Circuits, also known as the Unified Power Format (UPF), provides a consistent way to specify power implementation intent throughout the design process, including synthesis, physical implementation, and verification. This consistency makes it easier to perform 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 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. 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 in March 2009. This standard is supported by a large number of EDA companies, including Synopsys. Detailed information on the standard can be found in the IEEE 1801 specification itself, available from IEEE Xplore at http://ieeexplore.ieee.org, or from the UPF Standard 1.0, available from Accellera at http://www.accellera.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.

Power Intent Concepts

The UPF language provides a way to specify the power requirements of a design, but without specifying explicitly how those requirements are implemented. The language specifies how to create a power supply network to each design element, the behavior of supply nets with respect to each other, and how the logic functionality is extended to support dynamic power switching to design elements. It does not contain any placement or routing information. The UPF specification is separate from the RTL description of the design.

In the UPF language, a *power domain* is a group of elements in the design that share a common set of power supply needs. By default, all logic elements in a power domain use the same primary supply and primary ground. Other power supplies may optionally 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 *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.

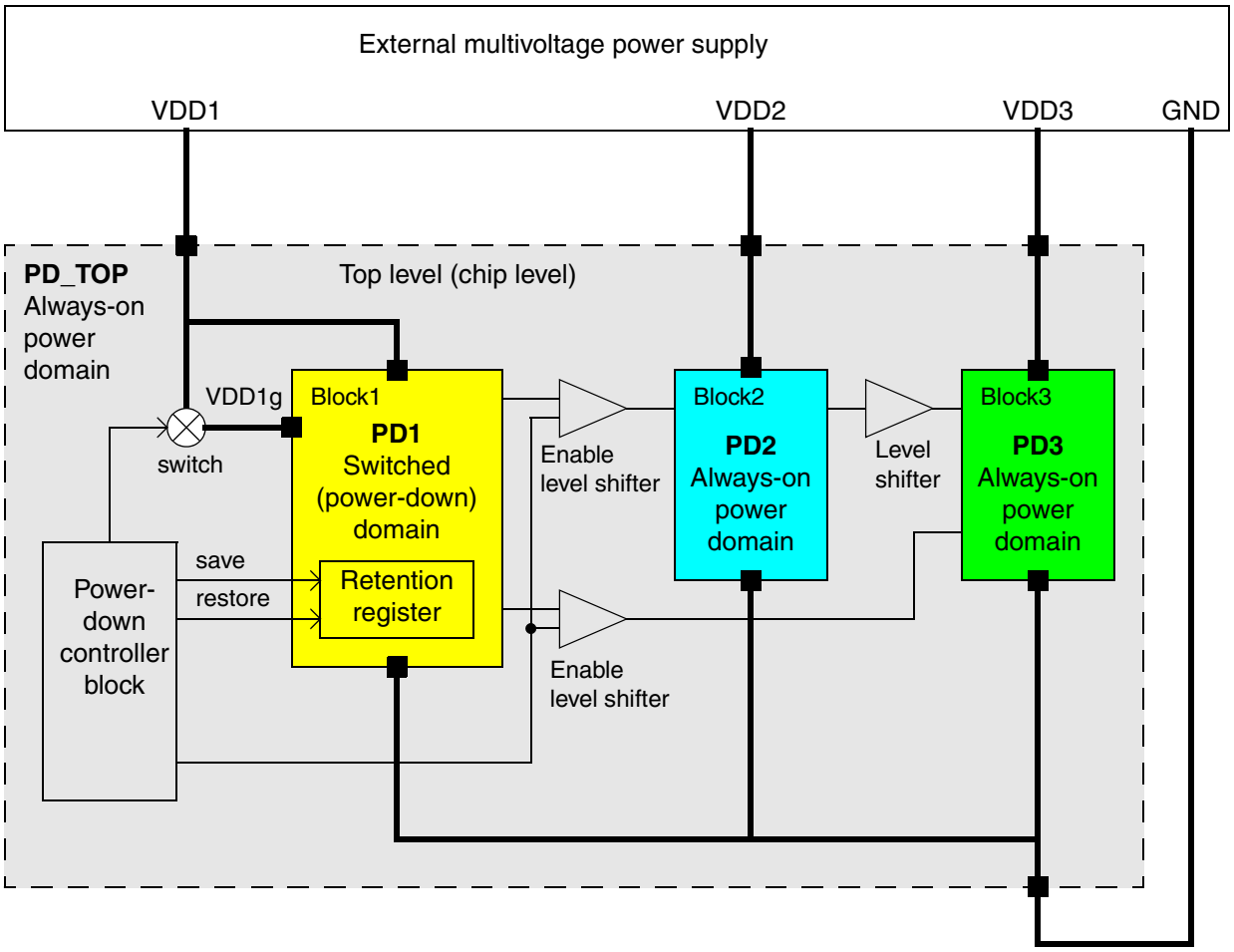
A *level shifter* must be present where a logic signal leaves one power domain and enters another at a substantially different supply voltage. The level shifter converts a signal from the voltage swing of the first domain to that of the second domain.

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 in the retention register while the primary supply of the domain is shut down.

The power network example shown in Figure 3-1 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.

Figure 3-1 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 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}
```

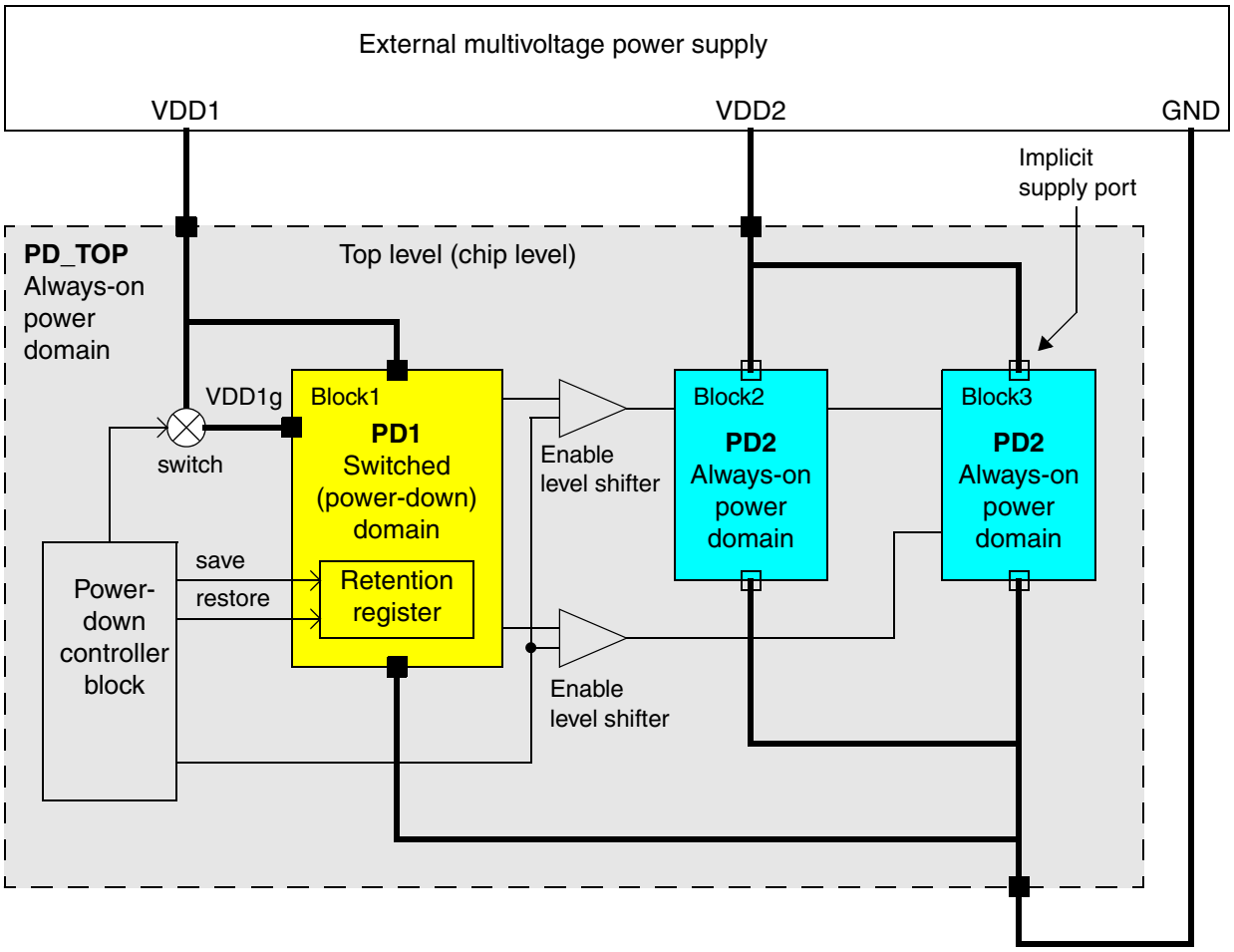
An alternative command entry method is to change the scope to Block1 using either the `set_scope` or `current_instance` command and using simple (not hierarchical) names in the commands:

```
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 3-2, 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.

Figure 3-2 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 diagram (Figure 3-1), 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 diagram (Figure 3-2), 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:

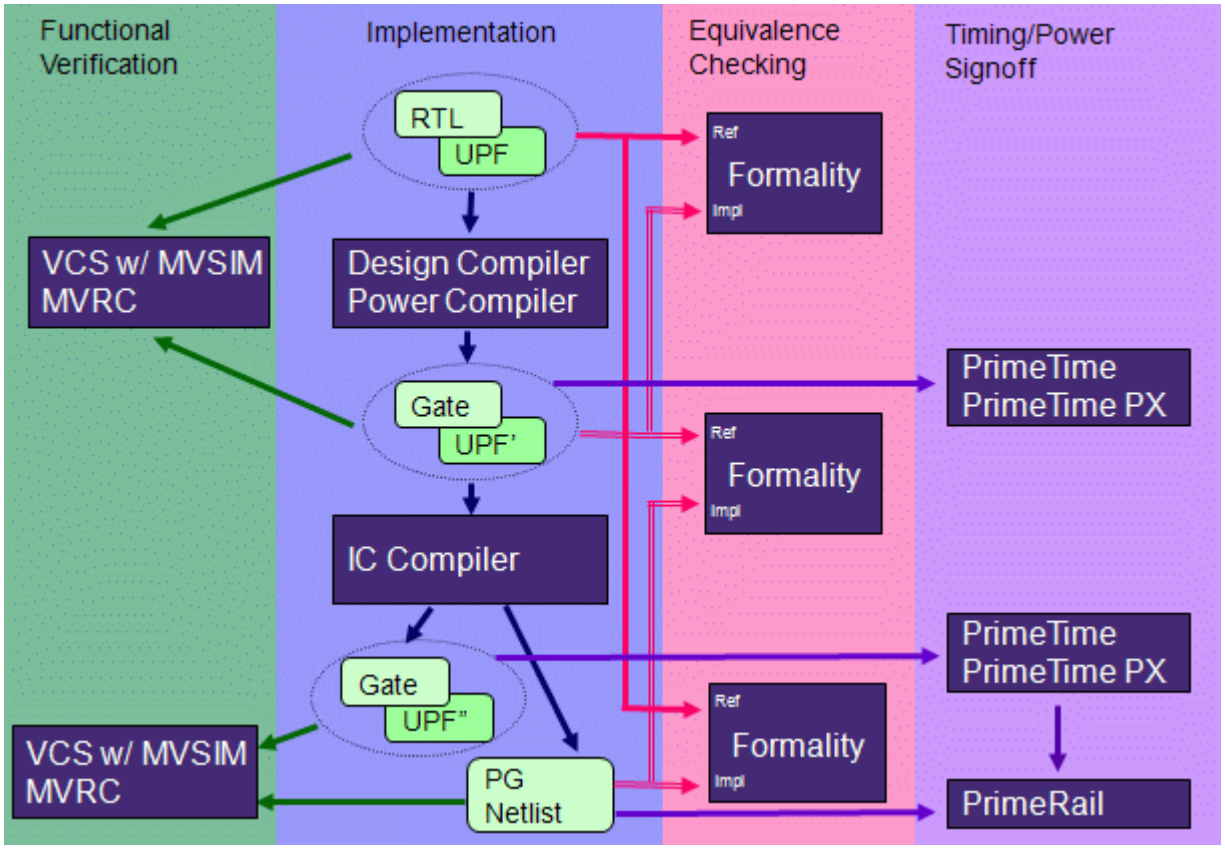
```
create_power_domain PD_TOP
set_scope Block1
create_power_domain PD1
set_scope ..
create_power_domain PD2 -elements {Block2 Block3}
```

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, IC Compiler might use larger or faster switching cells to provide power to the block that draws a larger current.

Synopsys Low-Power Flow

The Synopsys low-power synthesis, implementation, and verification flow is shown in Figure 3-3. 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 3-3 Low-Power Flow With Synopsys Tools



Design Compiler 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.

IC Compiler 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 Formality, and timing and power verification with PrimeTime, PrimeTime PX, and PrimeRail.

The VCS simulator and MVSIM 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 and MVSIM co-simulate 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. MVRC checks for adherence to multivoltage rules and reports any problems related to power connectivity, power architecture, or power intent consistency.

PrimeTime reads the gate-level netlist from Design Compiler or IC Compiler 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. PrimeTime does not modify the power domain description in any structural or functional way, so it does not write out any UPF commands.

UPF Commands in Synopsys Tools

Synopsys tools currently support a large subset of the commands in the version 1.0 standard. In addition, they support some UPF-like power intent commands that are not part of the official standard. Certain UPF commands are supported by some Synopsys tools but not others. For example, the synthesis and physical implementation tools generate new UPF commands, so they support the `save_upf` command, whereas the static timing and verification tools do not generate any UPF commands, so they do not support the `save_upf` command.

Table 3-1 lists the major UPF commands and shows whether they are supported by the following Synopsys tools: Synthesis (Design Compiler, Power Compiler, and DFT Compiler), IC Compiler, PrimeTime and PrimeTime PX, Formality, PrimeRail, VCS-MVSIM, and MVRC. A “yes” entry in the table under the tool name indicates support for the UPF commands listed in the leftmost column. A blank space indicates that the commands are not supported by the tool because they do not apply to the functions of the tools. The word “partial” next to a UPF command means that some options of the UPF-standard command are supported while others are not.

Table 3-1 Support for UPF Commands in Synopsys Tools

UPF commands	Synthesis DC PC DFT	IC Compiler	PT & PT PX	Formality	Prime-Rail	VCS-MVSIM	MVRC
Load/save/scope UPF:							
upf_version	--	--	yes	yes	yes	--	--
set_scope	yes	yes	yes	yes	yes	yes	yes
load_upf (partial)	yes	yes	yes	yes	yes	yes	yes
save_upf (partial)	yes	yes	--	--	--	--	--
Basic power:							
create_power_domain	yes	yes	yes	yes	yes	yes	yes
create_supply_net	yes	yes	yes	yes	yes	yes	yes
create_supply_port	yes	yes	yes	yes	yes	yes	yes
set_domain_supply_net	yes	yes	yes	yes	yes	yes	yes
connect_supply_net (partial)	yes	yes	yes	yes	yes	yes	yes
create_power_switch (partial)	yes	yes	yes	yes	yes	yes	yes
map_power_switch	--	yes	--	--	--	--	yes
Level shifter:							
set_level_shifter	yes	yes	--	--	--	yes	yes
map_level_shifter_cell	yes	yes	--	yes	--	yes	yes
name_format	yes	yes	--	yes	--	yes	yes
Isolation:							
set_isolation	yes	yes	yes	yes	--	yes	yes
set_isolation_control	yes	yes	yes	yes	--	yes	yes
map_isolation_cell	yes	yes	--	yes	--	yes	yes
name_format	yes	yes	--	yes	--	--	--
Retention:							
set_retention	yes	yes	yes	yes	--	yes	yes
set_retention_control	yes	yes	yes	yes	--	yes	yes
map_retention_cell (partial)	yes	yes	--	----	--	----	yes
Power state table:							
add_port_state	yes	yes	--	yes	--	yes	yes
add_pst_state	yes	yes	--	yes	--	yes	yes
create_pst	yes	yes	--	yes	--	yes	yes
Verification extension:							
set_design_top	--	--	yes	--	--	yes	yes

The following subsections describe the UPF commands supported by Synopsys tools. These descriptions are intended to provide an overview of how the commands work in the Synopsys flow. They are not intended to serve as a comprehensive command reference. Only the commands and command options supported by Synopsys tools are shown. The official IEEE 1801 (UPF) specification has some additional commands and command options that are not supported by Synopsys tools. Conversely, some Synopsys commands offer features that are not a part of the IEEE 1801 (UPF) specification.

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

Load/Save/Scope Commands

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

upf_version

```
upf_version string
```

The `upf_version` command specifies the UPF version number for which the subsequent UPF syntax is intended or returns the current UPF version number. As of the current release, only UPF version 1.0 is supported.

set_scope

```
set_scope [instance]
```

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

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

load_upf

```
load_upf upf_file_name
  [-scope instance_name]
  [-version upf_version]
```

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

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

The `-version` option specifies the UPF version number used to interpret the UPF commands.

Design Compiler lets you remove loaded UPF commands with the `remove_upf` command.

save_upf

```
save_upf upf_file_name
```

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

Only Design Compiler and IC Compiler make changes to the power design intent, so they are the only Synopsys tools that use the `save_upf` command. For example, Design Compiler inserts isolation cells and level shifter cells, and IC Compiler inserts power switches. After making these types of changes, the tool can write out a new set of UPF commands that include the added power features. The generated UPF file can then be used by downstream tools. Other Synopsys tools such as VCS, Formality, PrimeTime, and PrimeTime PX do not change the power design intent, and therefore do not support the `save_upf` command.

Basic Power Commands

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

create_power_domain

```
create_power_domain domain_name
  [-elements list]
  [-include_scope]
  [-scope instance_name]
```

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

The `-elements` option specifies a list of design elements that are assigned to the power domain (the extent of the power domain). The `-include_scope` option causes the entire hierarchical level of the domain to be included in the extent of the domain. If neither `-elements` nor `-include_scope` is used, the power domain includes all elements in the current scope (including lower-level elements) not already assigned to power domains by previous `create_power_domain` commands.

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

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

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

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

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

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

create_supply_net

```
create_supply_net net_name
  -domain domain_name
  [-reuse]
  [-resolve {unresolved|parallel}]
```

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

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

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

The `-resolve` option specifies how the state and voltage of a supply net are resolved when the supply net is driven by one or more power switches. You can specify either `unresolved` (single driver allowed) or `parallel` (multiple drivers allowed).

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

create_supply_port

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

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

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

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

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

```
create_supply_port Block1/VDD1P
```

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

set_domain_supply_net

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

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

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

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

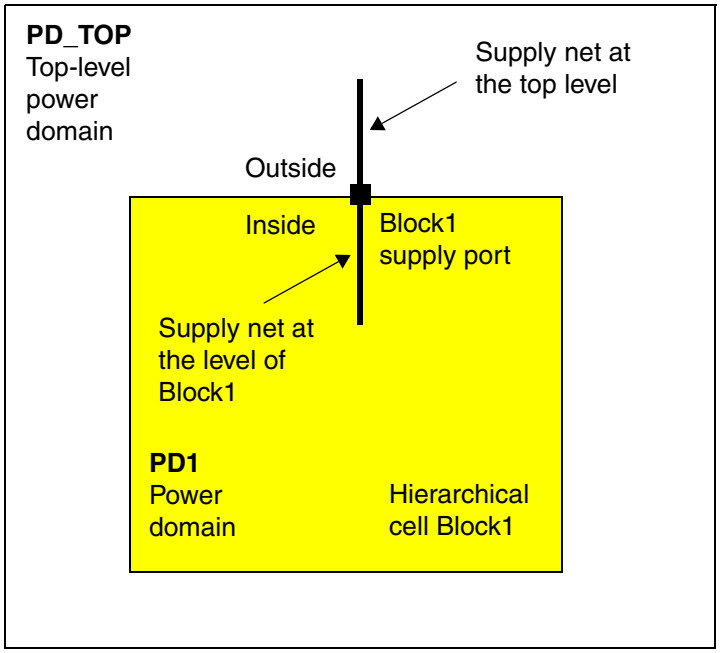
connect_supply_net

```
connect_supply_net supply_net_name
  -ports list
```

The `connect_supply_net` command specifies an explicit connection of a supply net to a list of supply ports, thereby overriding any implicit connections that might otherwise apply. The command specifies the name of an existing supply net and lists the supply ports to be connected to the supply net.

Each supply port has two ends: an inside end and an outside end. A supply net from the logical hierarchy above a supply port can connect only to the outside end of the supply port, and a supply net at the scope of the supply port can connect only to the inside end of the supply port, as shown in Figure 3-4. Supply nets existing below the logical hierarchy of the supply port can connect to the supply port only through supply pins at each hierarchical level.

Figure 3-4 Supply Net Connections to the Two Ends of a Supply Port

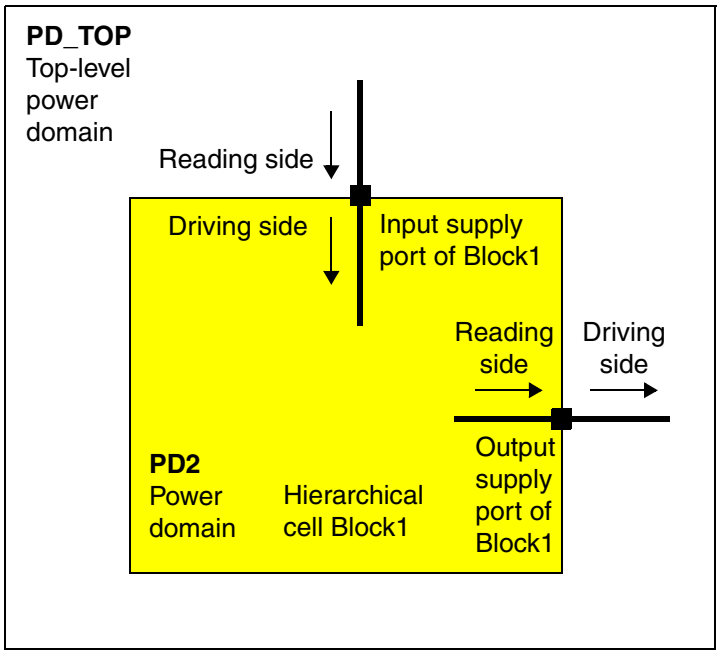


A supply port can be connected only once at each end. This restriction ensures a unique mapping from a net segment in the PG netlist to the corresponding supply net. To make the outside and inside connections, with the current scope set to the top level, you could use commands similar to the following:

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

VCS and Formality enforce restrictions on the “direction” of power flow. An output port “drives” its external net and “reads” its internal net; the “driving side” is the external net and the “reading side” is the internal net. On the other hand, an input port “drives” its internal net and “reads” its external net; the “driving side” is the internal net and the “reading side” is the external net. See Figure 3-5. You cannot connect more than one driver to a supply net or reuse an existing net with multiple drivers, unless that net has a resolution mechanism.

Figure 3-5 Power Flow Direction



create_power_switch

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

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

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

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

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

A power switch can optionally have one or more acknowledge output ports that indicate the state of the switch. If a Boolean function is specified by the `-ack_port` option, the result of the Boolean function is driven on the acknowledge port. Otherwise, the value 1 or 0 is driven on the acknowledge port when the switch is turned on or off, respectively, with a delay specified by the `-ack_delay` option. If the delay is specified as an integer, the time unit is interpreted as the precision of the simulator. The default delay value is zero. Any `-ack_port` or `-on_state` Boolean expressions are written in SystemVerilog syntax.

Design Compiler does not use the information in the `create_power_switch` command. Although it records the `-on_state`, `-off_state`, `-ack_delay`, and `-ack_port` Boolean function settings, it does not use this information for synthesis. Additional non-UPF commands in IC Compiler may be required to specify the mapping. Because verification tools such as VCS do not support any mapping commands, they use the information embedded in the `-on_state`, `-off_state`, and `-ack_port` settings to determine functionality.

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

The following is a simple power switch definition:

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

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

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

map_power_switch

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

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

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

For a single-input, single-output, single-control switch, the power pin names and signal pin names in the library cells need not match the virtual port names declared in the `create_power_switch` command. However, for a multiple-control switch, additional non-UPF IC Compiler commands may be needed to perform sophisticated tasks such as

mapping to a daisy chain of switches. After mapping, IC Compiler might generate additional `connect_supply_net` commands in its output UPF to represent the connections between the supply nets and the power pins on the mapped switches.

Level Shifter Commands

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

set_level_shifter

```
set_level_shifter strategy_name
  -domain domain_name
  [-elements port_pin_list]
  [-applies_to inputs | outputs | both]
  [-threshold float]
  [-rule low_to_high | high_to_low | both]
  [-location self | parent | fanout | automatic]
  [-no_shift]
```

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

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

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

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

- `self` – The level shifter cell is placed inside the model/cell being shifted.
- `parent` – The level shifter cell is placed in the parent of the cell /model being shifted.

- `fanout` – Level shifters are placed at all fanout locations (sinks) of the port being shifted.
- `automatic` – The implementation tool is free to choose the appropriate location. This is the default behavior.

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

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

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

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

Design Compiler and IC Compiler ensure that level shifter insertion does not alter the logical function of the design. The generated gate-level netlist, with exception of supply net connections for level shifters, is logically equivalent to the input RTL. For this reason, Formality and VCS do not need to consider `set_level_shifter` commands.

map_level_shifter_cell

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

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

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

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

name_format

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

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

If the generated name conflicts with another previously defined name in the same name space, the generated name is further extended by an underscore character, followed by a positive integer. An empty string is a valid value for any prefix or suffix option. When the prefix and suffix are both an empty string, only the underscore and number string combination are used as a suffix to create the new instance name.

Isolation Commands

The isolation commands specify the strategy for inserting isolation cells at the outputs of switched (power-down) domains.

set_isolation

```
set_isolation isolation_strategy_name
  -domain power_domain
  [-isolation_power_net isolation_power_net]
  [-isolation_ground_net isolation_ground_net]
  [-clamp_value  0 | 1 | z | latch]
  [-applies_to inputs | outputs | both]
  [-elements objects]
  [-no_isolation]
```

The `set_isolation` command specifies the isolation strategy for a power domain and the elements in that domain on which the strategy is applied. An isolation strategy includes specification of the enable signal net, the clamp value, and the location (inputs, outputs, or both).

At a minimum, either `-isolation_power_net` or `-isolation_ground_net` must be specified unless `-no_isolation` is used. If only `-isolation_power_net` is specified, the primary ground net is used as the isolation ground supply. If only `-isolation_ground_net` is specified, the primary supply net is used as the isolation power supply. If both are used, they specify the supply nets to use as the isolation power and ground nets. The isolation

power and ground nets are automatically connected to the implicit isolation circuit. If `-no_isolation` is specified, it means that the elements in the `-elements` list will not be isolated.

The `-elements` option can be used to specify the elements to isolate in cases where there are multiple isolation strategies within a given power domain. The listed elements (input/output ports on the domain boundary) must be within the specified power domain. If `-elements` directly specifies a port by name (not implicitly, by specifying the port's instance or an ancestor of that instance), then the isolation strategy applies to that port regardless of whether that port's mode matches the one specified by the `-applies_to` option. Without the `-elements` option, the isolation strategy applies to the whole power domain.

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

The `-clamp_value` option specifies the constant value of the isolation output: `0`, `1`, latch, or `Z`. The `latch` setting causes the value of the of the non-isolated port to be latched when the isolation signal becomes active.

The `-applies_to` option specifies the parts of the power domain that are isolated: inputs, outputs, or both. The default is outputs.

Although the power state table can potentially reduce the number of isolation cells required, isolation synthesis (pertaining to implementation and verification tools) for the Synopsys A-2007.12 release is entirely based on directives set with the `set_isolation` and `set_isolation_control` commands.

Certain types of optimization can be performed on isolation circuits by the implementation tool as long as the functionality is not affected. For example, suppose you have two blocks, A and B, with signals going from A to B. You specify output isolation in A in the parent and input isolation on B in the parent. If the strategy results in two back-to-back isolation cells with no fanout in between, the implementation tools can merge the isolation cells.

Design Compiler requires the isolation power and ground nets to operate at the same voltages as the primary and ground nets of the power domain where the isolation cells will be located, respectively.

The isolation strategies have the following levels of precedence, from highest to lowest, irrespective of the order in which they are executed:

```
set_isolation -domain -elements (with optional -applies_to)
set_isolation -domain -applies_to <input/output>
set_isolation -domain (with optional -applies_to both)
```

Every isolation strategy defined by a `set_isolation` command must have a corresponding `set_isolation_control` command unless the strategy is `-no_isolation`.

set_isolation_control

```
set_isolation_control isolation_strategy_name
  -domain power_domain
  -isolation_signal isolation_signal
  [-isolation_sense  0 | 1 ]
  [-location self | parent ]
```

The `set_isolation_control` command allows the specification of the isolation control signal and the logical sense of that signal. The command identifies an existing isolation strategy and specifies the isolation control signal for that strategy.

The `-location` option defines where the isolation cells are placed in the logic hierarchy: either `self` (the default) to place isolation cells inside the model/cell being isolated or `parent` to place the them in the parent of the cell /model being isolated.

The `-isolation_sense` option specifies the logic state of the isolation control signal, either 0 or 1, that places isolation cells in the isolation mode. The default is 1.

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

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

map_isolation_cell

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

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

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

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

name_format

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

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

If the generated name conflicts with another previously defined name in the same name space, the generated name is further extended by an underscore character followed by a positive integer. An empty string is a valid value for any prefix or suffix option. When the prefix and suffix are both an empty string, only the underscore and number string combination are used as a suffix to create the new instance name.

Retention Commands

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

set_retention

```
set_retention retention_strategy_name
  -domain power_domain
  -retention_power_net retention_power_net
  -retention_ground_net retention_ground_net
  [-elements objects]
```

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

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

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

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

```
set_retention –domain –elements
set_retention –domain
```

Design Compiler requires the isolation power and ground nets to operate at the same voltages as the primary and ground nets of the power domain where the retention cells will be located, respectively.

Every retention strategy defined by a `set_retention` command must have a corresponding `set_retention_control` command.

set_retention_control

```
set_retention_control retention_strategy_name
  -domain power_domain
  -save_signal {save_signal high | low}
  -restore_signal {restore_signal high | low}
```

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

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

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

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

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

map_retention_cell

```
map_retention_cell retention_strategy_name
  -domain power_domain
  [-lib_cells lib_cells]
  [-lib_cell_type lib_cell_type]
  [-elements objects]
```

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

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

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

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

Power State Table Commands

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

create_pst

```
create_pst table_name
  -supplies list
```

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

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

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

add_port_state

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

`*`Asterisk indicates an item that can be repeated in the command.

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

You can specify a nominal voltage or a range of voltages for a state. However, a port might be in any of the defined states and can take any voltage within a defined range, so a tool cannot determine the voltage to use for optimizing or analyzing the design from the power

state table alone. In Design Compiler, IC Compiler, and PrimeTime, you use the `set_voltage` command (a non-UPF command) to specify the operating voltage for optimization or analysis.

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

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

A power switch supply port is considered a supply port because it is connected by a supply net, so it can be specified as the supply port in the `add_port_state` command. Similarly, an RTL port can be made a supply port by this command or by the `connect_supply_net` command. Note that supply states specified at different supply ports are shared within a group of supply nets and supply ports directly connected together. However, this sharing does not happen across a power switch.

add_pst_state

```
add_pst_state state_name
  -pst table_name
  -state supply_states
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Verification Extension

The verification extension command is not a part of the UPF 1.0 specification. It is provided to support low-power checking by simulation and verification tools.

set_design_top

```
set_design_top instance
```

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

Power Supply Reporting and Collection Commands

In addition to supporting most of the commands in the UPF 1.0 standard, many Synopsys tools have non-UPF commands that report power-related objects in the design. Other commands are available that create collections of such objects. The reporting and collection commands are not part of the UPF 1.0 specification and vary from tool to tool, depending on the need for the commands in each tool. Some of these commands are shared by multiple tools and have similar or identical operation in different tools.

In many tools, the commands `report_power_domain`, `report_power_switch`, `report_supply_net`, and `report_supply_port` report the power-related objects in the design previously defined by UPF commands. Each command reports the objects existing in the design and the characteristics of the object. For example, the `report_supply_net` command lists the supply nets and shows the name, scope, power domains, supply ports, PG pins, voltage, and resolution status of each supply net.

The commands `get_power_domains`, `get_power_switches`, `get_supply_nets`, and `get_supply_ports` are commands that create collections of power-related objects in the design. You can then use a collection for custom reporting or as input to a script.

For more information about a particular reporting or collection command, see its man page in the applicable tool.

Low-Power Intent in Earlier Releases

Support for UPF commands and low-power infrastructure began with the A-2007.12 release. Many of the low-power features supported in the A-2007.12 release were also supported in previous releases, but proprietary Synopsys syntax was used instead of UPF syntax. The older syntax and features were still supported in the A-2007.12 release.

In some tools, only one set of low-power commands can be supported at any given time. As a result, the tool must be invoked or placed in either the UPF or non-UPF mode to support the commands and infrastructure of that mode.

Starting with the B-2008.09 release, Design Compiler and IC Compiler operate in the UPF mode by default. In earlier releases, it was necessary to explicitly specify the UPF mode upon startup using the `-upf_mode` option with the tool invocation command. In PrimeTime, the variable `power_domains_compatibility` can be set to `true` to support the use of older power intent commands.

Information about the older low-power flow commands is available in the documentation for each tool.

4

UPF Script Examples

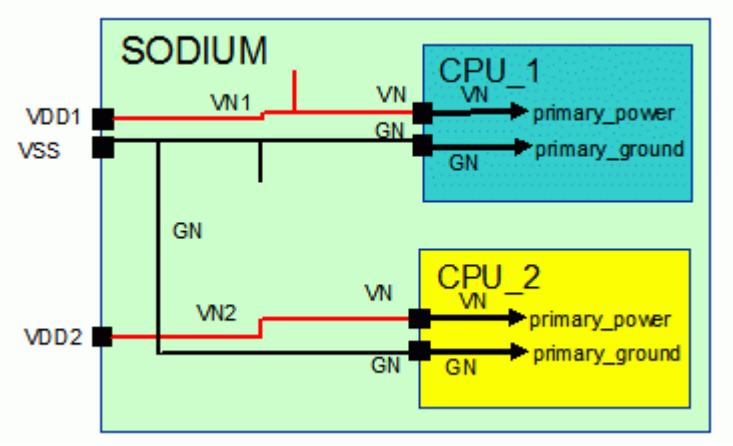
The following low-power flow examples demonstrate the UPF syntax used for specifying power intent.

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

Simple Multivoltage Design

The simple multivoltage chip design shown in Figure 4-1 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.

Figure 4-1 Simple Mulitivoltage 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
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}
 ...

# 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
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.
```

Top-Down Power Intent Specification

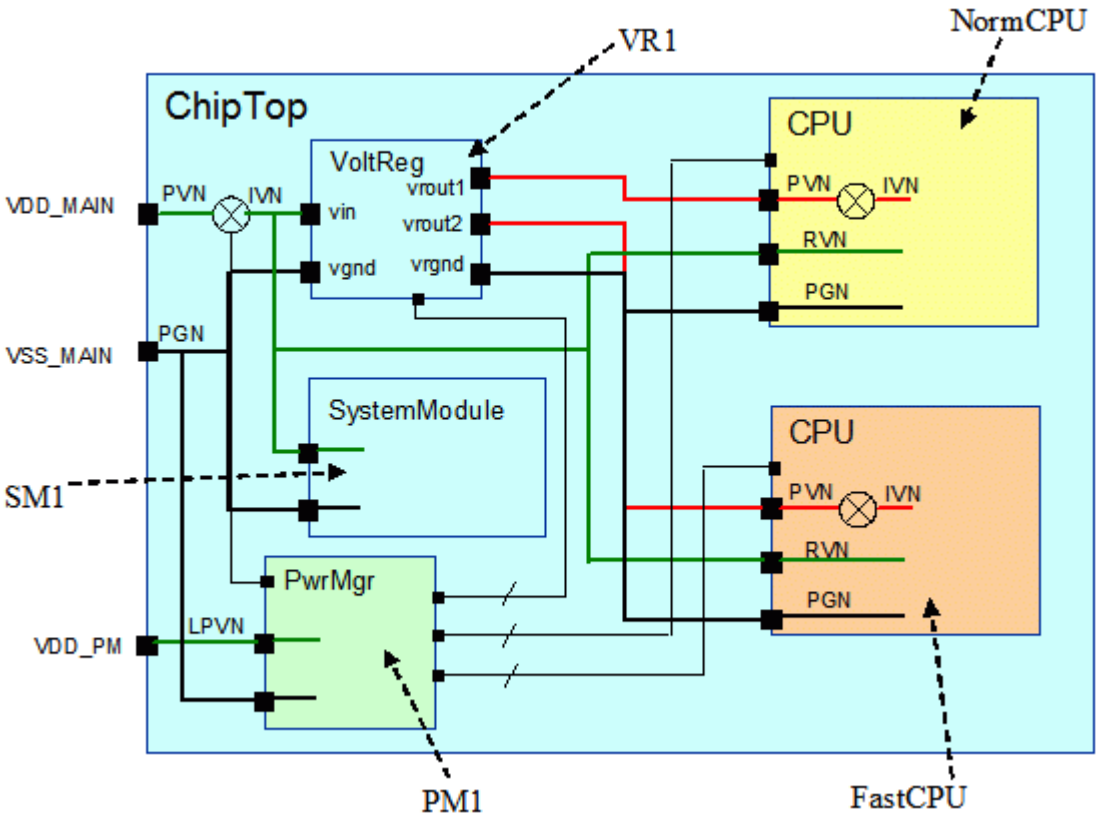
If the chip is designed from the top down, the power intent of the design can be specified directly from the top level, if desired, as demonstrated by the following example:

```
create_power_domain PD_CPU_1 -elements {CPU_1}
create_power_domain PD_CPU_2 -elements {CPU_2}
create_power_domain PD_SODIUM
create_supply_port VDD1
create_supply_port VDD2
create_supply_port VSS
create_supply_net VN1 -domain PD_CPU_1
create_supply_net VN1 -domain PD_SODIUM -reuse
connect_supply_net VN1 -ports {VDD1}
create_supply_net VN2 -domain PD_CPU_2
connect_supply_net VN2 -ports {VDD2}
create_supply_net GN -domain PD_CPU_1
create_supply_net GN -domain PD_CPU_2 -reuse
create_supply_net GN -domain PD_SODIUM -reuse
connect_supply_net GN -ports {VSS}
set_domain_supply_net PD_CPU_1 \
 -primary_power_net VN1 -primary_ground_net GN
set_domain_supply_net PD_CPU_2 \
 -primary_power_net VN2 -primary_ground_net GN
set_domain_supply_net PD_SODIUM \
 -primary_power_net VN1 -primary_ground_net GN
```

Switched Power Supply Example

The chip design shown in Figure 4-2 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 4-2 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 \
  -isolatio_signal cpu_iso \
  -isolation_sense low \
  -location self
```

This is the top-level power intent specification, chiptop.upf:

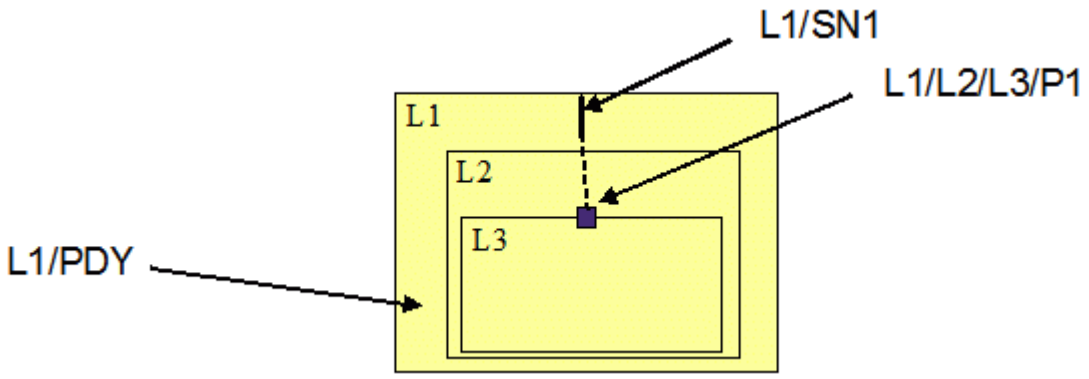
```
set_scope FastCPU
load_upf cpu.upf
set_scope ../NormCPU
load_upf cpu.upf
set_scope # doing this make chiptop not reusable.

# 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}

# 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 IP's
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_net Command

The `get_supply_net` command provides the ability to find the logical net name associated with a supply net in the specified domain for the specified scope. Figure 4-3 and the following code example demonstrate how to use the `get_supply_net` command to make a supply net connection across hierarchical levels.

Figure 4-3 Hierarchical Supply Connection Using get_supply_net



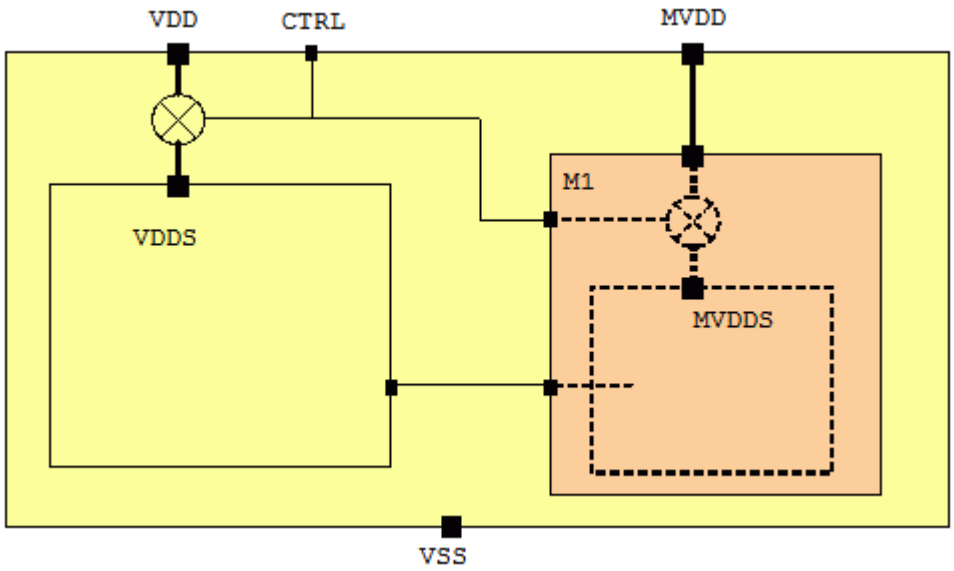

```
create_power_domain PDY -scope L1 -elements {L1}
create_supply_net SN1 -domain L1/PDY
create_supply_port L1/L2/L3/P1
set n [get_supply_net SN1 -domain L1/PDY -scope L1/L2]
connect_supply_net $n -ports {L1/L2/L3/P1}
# an alternative to connect_supply_net L1/SN1 -ports {L1/L2/L3/P1}
```

Power Switching States for a Macro Cell

In the power intent example shown in Figure 4-4, 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.

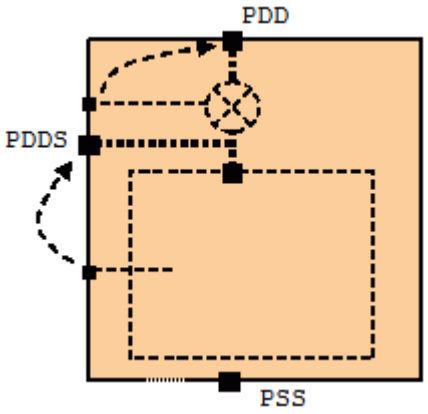
Figure 4-4 Macro Cell With Power Switch



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 4-5 schematically.

Figure 4-5 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
```

5

Tool-Specific Usage Recommendations

This chapter provides information on 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 may be different from what is described in this book. Refer to the individual product release notes for the most current information.

Power-Aware Verification Using MVSIM and MVRC

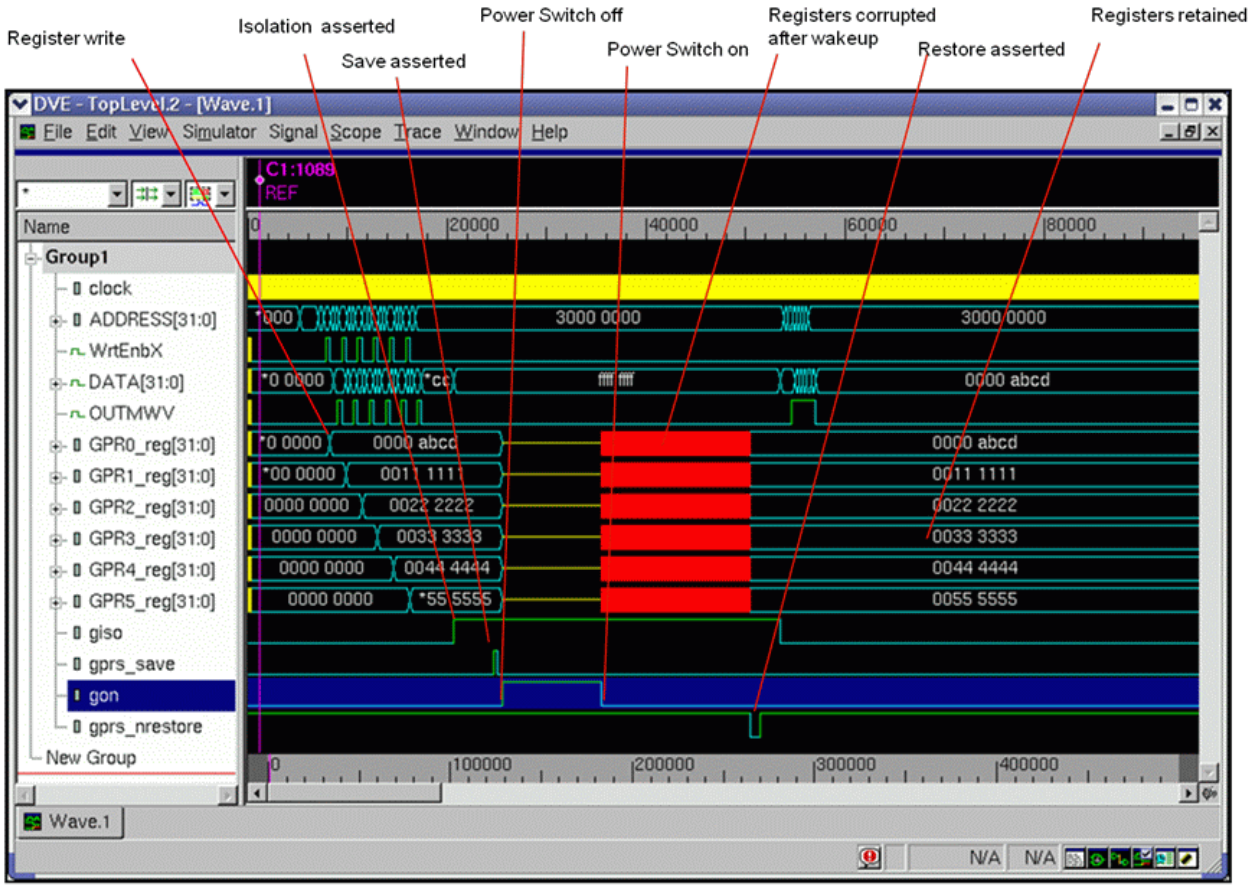
In the Synopsys functional verification flow, you can use VCS+MVSIM for multivoltage functional simulation and MVRC for static multivoltage rule checking. VCS and MVSIM co-simulate the design to verify the impact of voltage changes in a power-managed chip, allowing you to detect any low-power design issues accurately and reliably. MVRC checks for adherence to multivoltage rules and reports any problems related to power architecture, power intent consistency, and power connectivity.

A VCS+MVSIM simulation verifies the power management architecture, including the power-up and power-down sequence and the policies for retention, isolation, and level shifting. The assertions performed in the simulation are written to the log file, mvsim.log, as shown in the following example:

```
[MVSIM] INFO 5003: MVSIM initialized in PROTECTED mode.
[MVSIM] INFO 5001: Design database found. Loading database from apdb/mvdb/design.db.
[MVSIM] INFO 5001: Class Library database found. Loading database from
apdb/cdb/mvspec.db.
[MVSIM] INFO 5617: Net tb.vdd will trigger ON/OFF state of Power Network Element VDD.
[MVSIM] INFO 5617: Net tb.dut.gprs_on will trigger ON/OFF state of Power Network
Element
lp_dut/gprs_sw/gprs_on
[MVSIM] INFO 5001: Cross-over database found. Loading database from
apdb/mvdb/xover.db.
[MVSIM] INFO 5102: Value of Isolation Enable changed. Isolation Enable signal
tb.dut.inst_iso_out corresponding to Protection Policy 'dut/inst_iso_out' changed to
value = 1 at time = 0 ps.
[MVSIM] INFO 5706: Power Network Element VDD of kind PowerNet started with ON State.
[MVSIM] INFO 5706: Power Network Element VDDG of kind PowerNet started with ON State.
...
[MVSIM] INFO 5109: Simulation started with Island TB in mode ACTIVE
[MVSIM] INFO 5109: Simulation started with Island dut/GENPP in mode ACTIVE
[MVSIM] INFO 5109: Simulation started with Island dut/GPRS in mode ACTIVE
[MVSIM] INFO 5109: Simulation started with Island dut/INST in mode ACTIVE
[MVSIM] INFO 5109: Simulation started with Island dut/MULT in mode ACTIVE
...
```

The VCS+MVSIM GUI can display the simulation waveforms in an easily readable form. Figure 5-1 shows an example.

Figure 5-1 VCS+MVSIM Simulation Results in GUI



In this waveform display example, VCS+MVSIM simulates the following sequence of events:

1. Load six registers in a power-down block.
2. Assert the retention save signal.
3. Power down the block (isolate, gate clocks, and turn off power switch.
4. Power up the block (turn on power switch, remove clock gating, and remove isolation).
5. Restore the register values.

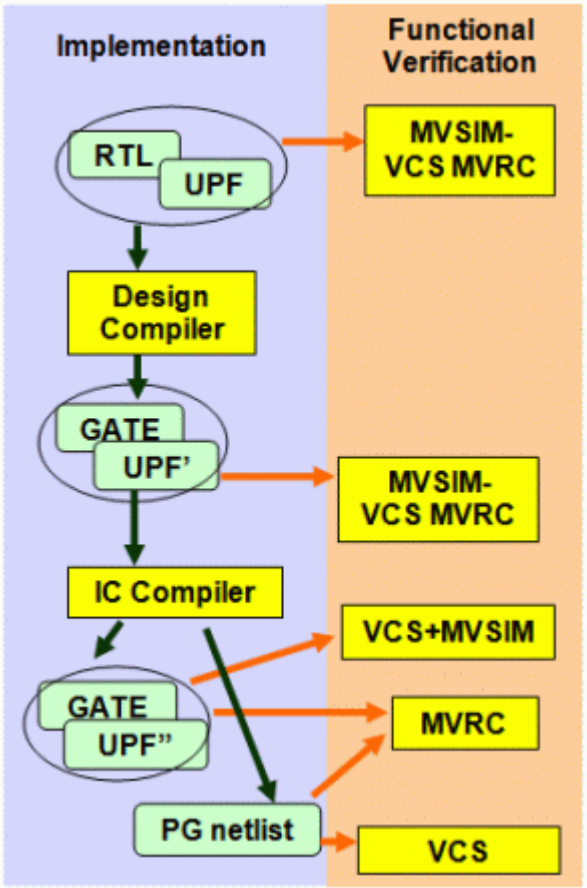
MVRC checks for adherence to multivoltage rules and power intent specification. The types of checking it performs are divided into the following categories:

- UPF critique/consistency/errors and power architecture: checks on isolation and level-shifting polices and elements versus requirements implied by the power state table, including 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.
- 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.

Simulation and rule checking can be performed at the stages of the design flow shown in Figure 5-2.

Figure 5-2 Power-Aware Verification Flow



These are the stages at which simulation and multivoltage rule checking are performed:

- Original RTL + UPF, before synthesis
- Gate-level netlist + UPF' produced by Design Compiler and Power Compiler

- Gate-level netlist + UPF'' and PG netlist produced by IC Compiler

Original RTL + UPF

After you prepare an RTL description and UPF file for a design, and before synthesis, you can use VCS+MVSIM and MVRC to verify the description for correct functionality and adherence to multivoltage rules. You perform this checking on the original RTL and UPF specification, in the absence of any multivoltage cells (isolation cells, level shifters, retention registers, or power switches) or power and ground (PG) connections in the RTL. For complete architectural rule checking, you should provide the control signals from the SDC file.

MVRC checks for adherence to multivoltage rules and power intent specification. It finds and reports unusual conditions such as missing level shifters, missing isolation cells, wrong type of interface cell (level shifter versus isolation cell), wrong polarity for isolation control signals, and incorrect power-up/power-down control signals.

This is a typical MVRC script used to check a design at the RTL + UPF level:

```
read_db ...
report_violation -architecture
report_violation
report_violation -signal control_signal_name
report_power_intent -mv_view report_power_intent -retention
```

Gate-Level Netlist + UPF' from Design Compiler

After synthesis with Design Compiler and Power Compiler, and before placement and routing, you can perform simulation and verification of the gate-level netlist and modified UPF file, UPF' (UPF prime) generated during synthesis. The analysis operates on a netlist with isolation cells and retention registers identified by pragmas, but without PG connections for logic cells and without power switches. For analysis at this stage, you provide the gate-level netlist, the UPF' file, the .db library models, the simulation models for standard cells, and PG-aware models for the isolation, level shifter, and retention cells.

The VCS+MVSIM simulation performs the same type of analysis as at the RTL+UPF stage, but using the gate-level netlist instead of RTL and including the retention, isolation, and level shifter cells inserted during synthesis. It verifies the power-aware behavior of the synthesized logic and inserted power-related cells. You can optionally perform simulations using SDF annotation to account for different signal propagation delays under various power-down modes.

Similarly, MVRC checks for adherence to multivoltage rules and power intent specification using the generated gate-level netlist with inserted power-related cells. It checks the implementation against the UPF-specified power intent for isolation, level-shifting, and retention functions.

This is a typical MVRC script used to check a design at the gate-level netlist + UPF' after synthesis by Design Compiler and Power Compiler:

```
read_db ...
report_protection -all
report_violation -architecture
report_violation
report_violation -signal <control signal name>
report_power_intent -mv_view
report_power_intent -retention
report_reachable -severity ERROR
```

Gate-Level Netlist + UPF" and PG Netlist from IC Compiler

After you complete placement and routing in IC Compiler, you can perform simulation and verification of the gate-level netlist and modified UPF file, UPF'' (UPF double-prime), generated for physical implementation. For analysis at this stage, you provide the gate-level netlist, the UPF'' file, the .db library models, and PG-aware models for all cells.

The PG-connected netlist has the power supply pins defined and connected as cell inputs and outputs, along with the logic pins, so it can be simulated with PG-aware simulation models by VCS without using MVSIM. The simulation performs the same type of analysis as at the earlier stages, including the power switches inserted by IC Compiler. It verifies the power-aware behavior of the synthesized logic and inserted power-related cells.

MVRC checks for adherence to multivoltage rules and power intent specification using the generated gate-level netlist with inserted power switches. It checks the implementation against the UPF-specified power intent for isolation, level-shifting, and retention functions. It also checks for always-on floating nets and pins, switch network functionality, and PG connectivity.

This is a typical MVRC script used to check a design after placement and routing by IC Compiler, using both the gate-level netlist + UPF'' and the PG-connected netlist:

```
read_phydb ...
report_protection_pg
report_retention_pg
report_switch_pg
report_special_cell_pg
report_connection_pg
remove_phydb
read_db ...
```

```
report_protection -all
report_violation -architecture
report_violation
report_violation -signal <control signal name>
report_power_intent -mv_view
report_power_intent -retention
```

Logic Synthesis Using Design Compiler

Several approaches can be used for the logic synthesis of multivoltage designs. Top-down synthesis is the simplest approach. In this case, 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 with a bottom-up approach.

Design Compiler, in combination with Power Compiler, 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. The topographical mode of Design Compiler 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.

When you start Design Compiler, the UPF mode is enabled by default. To invoke Design Compiler in non-UPF mode, you would use the `-non_upf_mode` switch when you invoke the tool. You can use the `shell_is_in_upf_mode` command to check the current mode of the Design Compiler shell. This command returns 1 when the shell is in UPF mode and 0 otherwise.

In UPF mode, Design Compiler reads, understands, and uses UPF-specified power intent. It ignores any `$` constructs (such as `$isolate` and `$retain`) used to specify power intent in earlier releases. You cannot use both $ constructs and UPF commands in the same session. A Power Compiler license is required for synthesis using UPF.

For synthesis, Design Compiler uses the RTL description of the design logic and the UPF description of the design power supply. The input UPF to Design Compiler is used for RTL simulation and formal equivalency checking, as well as for synthesis. After it completes synthesis, Design Compiler writes out a gate-level UPF description, which includes the original UPF read into Design Compiler, any additional UPF commands run in the Design Compiler session, and UPF commands the describe the explicit supply connections to special-purpose cells added during synthesis (isolation cells, level shifters, and retention registers). The UPF description generated by Design Compiler can be used by Formality, VCS, IC Compiler, and PrimeTime.

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 or compile_ultra` command.
5. Insert the scan chains using the `insert_dft` command.
6. Use the `save_upf` command to save the updated constraints in 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.

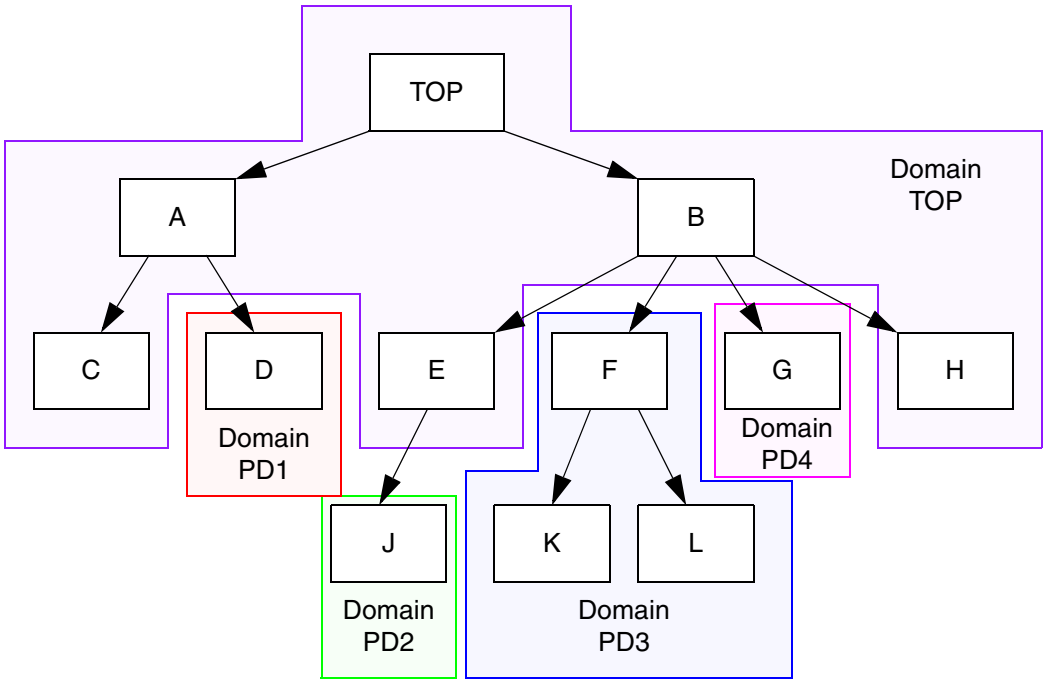
Power Domains and Hierarchy

Power domains defined for the design determine the connections of power nets and the implementation of power-down voltage areas. Power domains can be defined on hierarchical blocks. Nested power domains are allowed as long as each power domain is contiguous and completely enclosed within any higher-level power domain. Each hierarchical cell can belong to only one power domain. For example,

```
create_power_domain TOP
create_power_domain PD1 -elements {A/D}
create_power_domain PD2 -elements {B/E/J}
create_power_domain PD3 -elements {B/F}
create_power_domain PD4 -elements {B/G}
```

The hierarchy and power domain membership of this example are illustrated in Figure 5-3.

Figure 5-3 Power Domains and Hierarchy Example

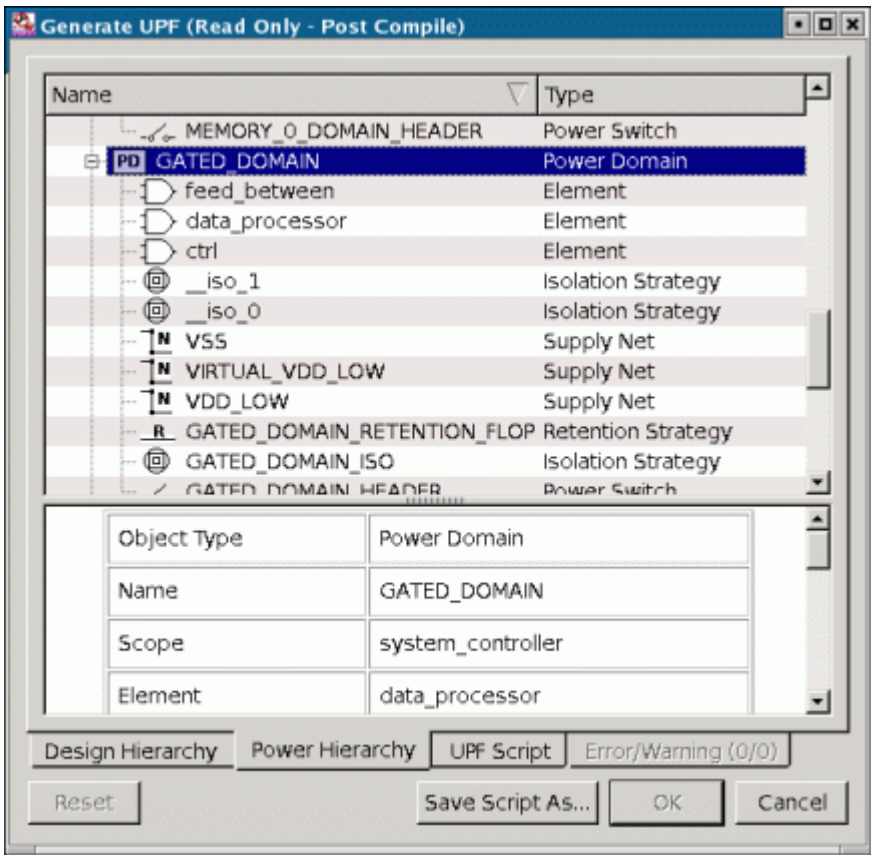


In this example, the first `create_power_domain` command assigns the TOP design to the power domain called TOP, which causes all the lower-level blocks to also belong to the same power domain. The subsequent `create_power_domain` commands assign some of the lower-level blocks to different power domains. Lower-level blocks not assigned to other power domains remain in the TOP power domain.

To find out about existing power domains that have been defined in the design, you can use the `report_power_domain` command, which generates a detailed report on the power domain characteristics, or the `get_power_domains` command, which returns a collection of power domains.

Another way to get power domain information is to use the Generate UPF dialog box in the Design Vision GUI. In Design Compiler, use the `gui_start` command to open Design Vision. In the Design Vision window, choose Power > Generate UPF. This opens the Generate UPF dialog box. Click the Power Hierarchy tab to view and examine the hierarchy of power domains and features within those power domains. See Figure 5-4 for an example.

Figure 5-4 Power Hierarchy View in Generate UPF Dialog Box



For information about using the Generate UPF dialog box, see the online help in Design Vision (Help > Online Help, under Using UPF Tools).

Specifying Operating Voltages

You can specify operating voltages of supply nets in the design with the `set_voltage` command. Parts of the design powered by these supply nets are timed and optimized based on the cell properties at the specified voltages. Process and temperature values are determined by the top-level operating condition.

For example, the following commands specify the target libraries, operating conditions, and supply net voltages for the design:

```
## Target Libraries
set target_library "HVT.db LVT.db SVT.db"

## Design operating condition name
```

```
set_operating_conditions WC09

## Set voltages on PN1 and PN2 supply nets
set_voltage 1.1 -object_list PN1
set_voltage 0.7 -object_list PN2
```

The voltages set on the supply nets must be consistent with the allowed voltages defined in the UPF power state table.

The `set_voltage` command allows minimum and maximum operating condition voltages to be specified for a supply net. If only one voltage is specified, it applies to the maximum operating condition. To specify voltages for both the maximum and minimum operating conditions, use the -min option, as in the following example:

```
set_voltage 0.86 -min 1.06 -object_list VDD
```

After execution of this command, Design Compiler uses a library cell characterized at 0.86 volts for max (worst corner) analysis and at 1.06 for min (best corner) analysis.

Instance-Based Targeting

Instance-based targeting, also known as target library subsetting, allows you to restrict the target library cells eligible for mapping and optimizing a particular logic hierarchy. The blocks in a design can be assigned different target libraries and can be optimized concurrently. The effect is that you can use specific libraries to optimize a specific logic hierarchy of the design.

You can use instance-based targeting to specify which library cells should be used in cases where the operating conditions and `set_voltage` alone are not enough to uniquely specify a particular library cell. The `set_target_library_subset` command specifies which library cells to use for a particular block or set of blocks in the design.

For example, in a dual-threshold technology, there might be two different libraries containing cells at two different threshold voltages, both operating under the same conditions and supply voltage. You might want to use high-threshold cells for most of the design for power savings but use low-threshold cells for block B to meet the higher speed requirements for that block. You would target the low-threshold library cells for block B as follows:

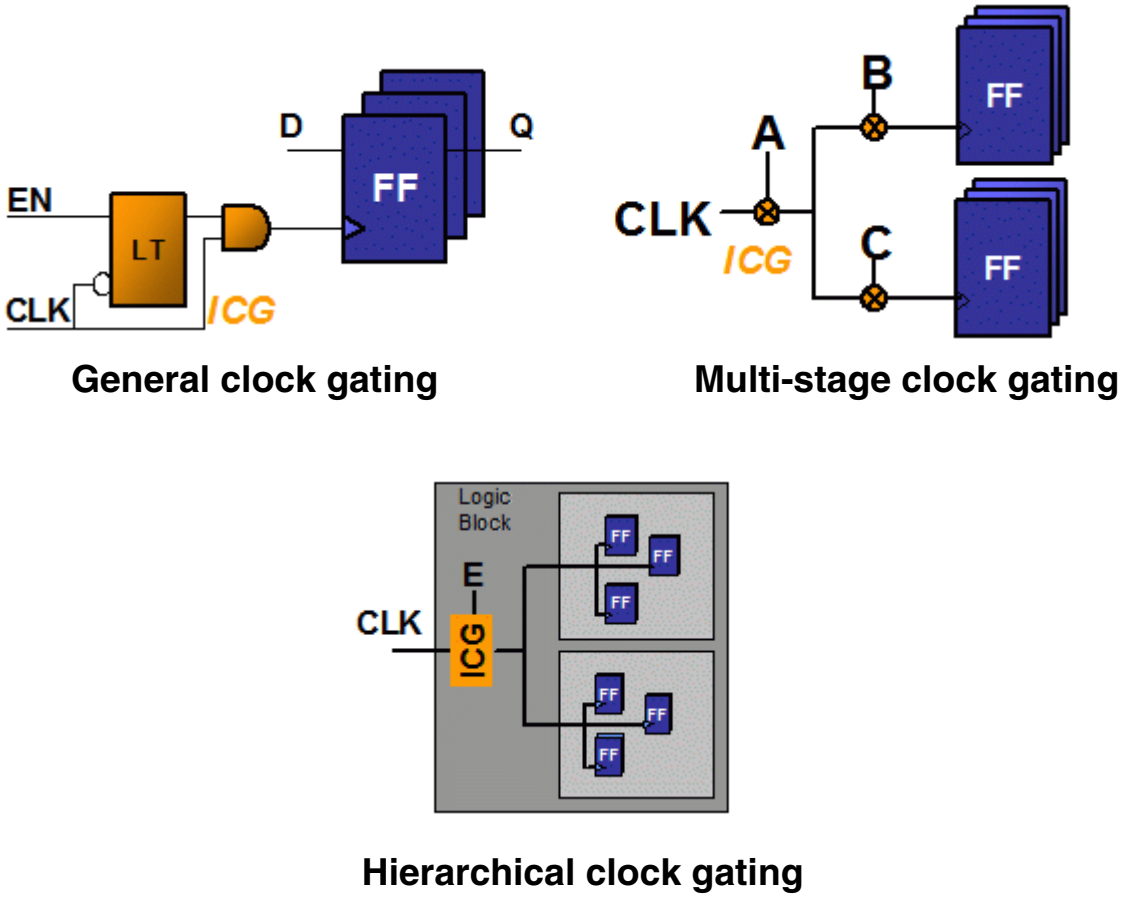
```
## Set target libraries: high Vt and low Vt
set target_library "HVT.db LVT.db"

## Use high-threshold cells at top level and below
set_target_library_subset "HVT.db" -top

## However, use low-threshold cells for instance B
set_target_library_subset "LVT.db" -object_list B

report_target_library_subset
```

Instance-based targeting is supported in both Design Compiler and IC Compiler, and applies to both logical and physical views.

Clock Gating

Apart from the standard style, in which one clock-gating cell is applied per register bank, clock gating can also be implemented in different modes, depending on the design architecture, to gain more dynamic power savings. Figure 5-5 shows three different clock-gating styles: general, multi-stage, and hierarchical.

Figure 5-5 Clock-Gating Styles



In multi-stage clock gating, further gating is done for the clock-gating cells which have common enable signals. As a result, more dynamic power can be saved. In the hierarchical clock-gating style, clock gating is done at the logic hierarchy level by gating hierarchical logic modules that share a common enable and the same clock group. This reduces the number of clock-gating cells in the design.

Power Compiler provides extensive features and control mechanisms that implement these clock-gating styles. The `set_clock_gating_style` command has options to specify clock-gating parameters such as integrated clock-gating cell usage, register bank width, and fanout. You can use the `-gate_clock` option of the `compile` or `compile_ultra` command to perform clock-gate insertion during compile. In that case, Power Compiler can perform clock-gate insertion on gate-level netlists as well as RTL and GTECH netlists. For example,

```
## SET CLOCK GATING STYLE
set_clock_gating_style  -sequential_cell latch \
                        -minimum_bitwidth 2 \
                        -num_stages 2 \
                        -control_point before \
                        -control_signal scan_enable \
                        -max_fanout 20 \
                        -positive_edge_logic integrated:CKLNQD8 \
                        -negative_edge_logic integrated:CKLHQD8

## PERFROM STANDARD CLOCK GATING
compile -gate_clock
```

In a multivoltage design, clock tree synthesis is performed separately for each voltage domain because the cell and net delays can be different for different domains. Design Compiler adds any necessary level shifters on the clock and clock-enable nets that cross voltage domain boundaries. Level shifter insertion on clock nets can be controlled by setting the `auto_insert_level_shifters_on_clocks` variable; see the man page for details.

Isolation, Level Shifter, and Retention Register Insertion

Design Compiler 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.

The size_only attribute is set on all inserted isolation cells, level shifters, and enable level shifters to prevent them from being optimized away. This ensures that the isolation and level-shifting functions between power domains are maintained throughout the flow.

Level shifters are the cells that function as the interface between power domains operating at different voltage levels. These cells ensure that the output transition of a driver can cause the receiving cell to switch even though the receiver is supplied with another voltage level.

While there are several level shifter implementations, they can be divided into two major classes: simple buffer-type level shifters and enable-type level shifters (often called enable level shifters). An enable level shifter, in addition to shifting the voltage level between input

and output, can maintain its output fixed at a steady value that is controlled by the enable signal, which allows the shut-off of its input pin. Design Compiler automatically inserts buffer-type level shifters in a compile operation.

Several options of the `set_level_shifter` command affect how buffer-type level shifters are added to the design. For example, the `-threshold` option establishes a threshold beyond which a voltage difference causes a level shifter to be inserted. The `-location` option specifies where the level shifters are placed in the logic hierarchy: `self`, `parent`, `fanout` (sinks), or `automatic` (the default; the tool chooses the best location for level shifter insertion).

After level shifters are inserted into the design, a dont_touch attribute is automatically set on the port-to-level-shifter net segment when the level shifter is within the block or on the level-shifter-to-port net segment when the level shifter is in the parent hierarchy. Also, a size_only attribute is set on the level shifter cell to prevent optimization from mapping the cell to some other logic. This means compile can perform sizing optimizations on level shifters.

Level shifters should also be added to the clock tree nets. The clock tree synthesis engine in IC Compiler assumes that all required level shifters have been added beforehand.

If level shifters are missing on particular nets, the timing engine maintains accurate timing analysis by automatically scaling transition times and delays according to the voltage on either side of the voltage interface.

Design Compiler inserts isolation cells where signals leave a power-down domain. After an isolation cell is inserted, a dont_touch attribute is automatically set on the net segment connecting the isolation cell power domain interface. This protection helps prevent optimizations from placing cells on that piece of net. Also, a size_only attribute is set on the isolation cell to prevent optimization from mapping the cell to some other equivalent logic.

Enable level shifters are used when both isolation and level shifting are required. This typically occurs when the power domain can be shut down and has a different voltage. The approach for inserting enable level shifters is similar to that of isolation cells. If enable level shifters are not available in the library, both a level shifter and an isolation cell need to be used on the same net. However, the dont_touch constraint (on the net segment from level shifter or isolation cell to the power domain boundary) prevents placement of both the special cells on same net segment. Therefore, the level shifter and the isolation cell need to be placed on the opposite sides of power domain interface.

Always-On Synthesis

The cells of the shutdown block that are on a control signal path must remain active during the powered-down phase. These signals include power-down/power-up control signals and retention register save signals.

Always-on synthesis is performed by inserting buffers and inverters along the control signal paths using either of the following types of cells:

- Standard, single-power cells that are placed in special always-on site rows within the shutdown block's voltage area
- Special-purpose, dual-power cells that are marked as always-on

Use the `set_always_on_strategy` command to specify either the single-power or dual-power strategy for a given power domain. This is the command syntax:

```
set_always_on_strategy
  -object_list list_of_domains
  -cell_type single_power | dual_power
```

The dual-power strategy uses special-purpose, always-on cells. Each of these cells has two power pins: a primary pin that is hooked up to the primary supply and a backup pin that is hooked up to a supply net that ensures the cell will remain powered when the rest of the domain is shut off. The cell also has an attribute called `always_on` which is set to true at the cell level. This type of cell is known as a dual-rail, always-on cell.

The single-power strategy uses standard, single-power cells that are placed in dedicated always-on site rows of the power domain's corresponding voltage area. These always-on sites are powered by the backup power supply. During optimization, the logic synthesis tool does not make any changes to these always-on site rows, so it leaves these cells on the always-on paths. During placement in IC Compiler, these cells are constrained to be placed in designated always-on site rows.

Compile

Compile performs logic synthesis on 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.

In Design Compiler topographical mode, all multivoltage features are completely supported. In topographical mode, the `create_voltage_area` command creates a voltage area for placing the cells associated with a particular power domain, like the same command in IC Compiler. This command completes the physical constraints linked to the multivoltage design that can be defined in Design Compiler topographical mode. Note that `compile_ultra`, the default synthesis command in Design Compiler topographical mode, automatically ungroups all design hierarchies to achieve better timing results. However, the logic hierarchies associated with power domains are not ungrouped and the power domain boundary information is therefore preserved.

Apart from the regular optimizations, compile also takes care of specific optimizations 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. The size_only attribute specified on level shifter and isolation cells allows compile to perform sizing optimization on these special cells.

Retention register inference and control port hookup operations are performed by compile. Compile maps user-specified registers in the shutdown power domain with retention registers. Compile synthesizes applicable registers with retention registers and hooks up the save and restore pins to control ports. Library retention cells should have specific attributes at cell and pin level to identify the retention cell and the “save” and “restore” control pins.

Leakage Power Optimization

Leakage power optimization using multi-threshold libraries results in leakage power savings. The `set_max_leakage_power` constraint causes compile to perform leakage power optimization.

The recommended methodology is to use all multiple-Vt libraries for logic synthesis in Design Compiler, and also for physical optimization in IC Compiler. The tool optimizes the design while providing equal emphasis for timing and power and making sure that the number of low-Vt cells used is kept to a minimum. This methodology gives comparable power numbers to that of two-pass leakage flow while maintaining good timing quality.

```
# Use all multi-Vth libraries
set_target_library "lvtwc.db svtwc.db_hvtwc.db"

# Set power options
set_max_leakage_power 0.0 mW

# Run compile
compile_ultra -scan
```

The leakage optimization flow is easy to use and gives excellent results.

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 may be needed.

The `check_mv_design` command checks multivoltage infrastructure of the design and issues error and warning messages about any violations found. This command should be used at various stages of the synthesis flow to check for multivoltage problems. This is the command syntax:

```
check_mv_design
  [-verbose]
  [-isolation]
  [-target_library_subset]
  [-opcond_mismatches]
  [-connection_rules]
  [-level_shifters]
  [-power_nets]
  [-max_messages message_count]
```

By default, the command checks for all types of multivoltage violations. You can restrict the scope of checking by using one or more of the following options:

- `-target_library_subset` – Checks for any inconsistency between the target library, target library subset, and operating conditions.
- `-isolation` – Checks for nets that need isolation but do not have it and nets that have isolation but do not need it.
- `-opcond_mismatches` – Checks for any inconsistency between the operating conditions set on a cell and the operating conditions at which the cell was characterized, or the operating conditions set on the parent cell.
- `-connection_rules` – Checks for violations of rules regarding always-on synthesis and pass gate connections such as an always-on net driven by a normal cell, an always-on cell driving a normal net, or an always-on net driving a pass gate.
- `-level_shifters` – Checks for violations involving level shifters such as nets that need level shifting but do not have it, nets that have a level shifter that is not needed, incorrect level-shifting voltages, and level shifter strategy violations.
- `-power_nets` – Checks for incorrect and missing power and ground connections to cells, which fail to match the type of connection derived from the characteristics of the power domain.

If the `-verbose` option is used, details of all violations are reported. The `-max_messages` option limits the number of violations reported.

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 impacts design timing and area quality of results. Therefore, the most common

requirement for multivoltage-aware DFT methodology is to make sure scan insertion is bounded by the power domains. In addition, when 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 Design Compiler, 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.

Scan Insertion Configuration

By default, DFT Compiler clusters the scan chains within a power domain. This clustering reduces the number of power domain crossings and therefore the number of level shifters and isolation cells needed in the scan path. However, if the design does not require the scan chains to be partitioned by power domains, the following commands can be used:

```
set_scan_configuration -power_domain_mixing true
set_scan_configuration -voltage_mixing true
```

When power domain mixing is enabled, a scan chain can span multiple power domains. Similarly, the voltage-mixing option allows a scan chain to mix cells from different voltage levels. Even if power domain mixing and voltage mixing are enabled, DFT Compiler attempts to keep the scan chains within each power domain as much as possible, thus minimizing the number of crossings to other power domains.

For the `insert_dft` command to properly insert level shifters and isolation cells, you must specify the level shifter and isolation cell strategies on a power-domain basis, even if you have specified similar strategies on blocks and individual ports.

Another way to specify the scan chains is to group the registers. DFT Compiler offers grouping features that allow specific selection of all the registers to be inserted in the chains. This is not an automatic feature, but it gives you full control of scan insertion in all power domains. The `set_scan_group` command can be used to group all the registers in a particular power domain that should be clustered:

```
set_scan_group scan_group_name -include_elements [list sequential_cells]
```

You can use the `set_scan_path` command to restrict the cells to a particular scan chain, You can specify either a list of cells or scan groups, or you can identify cells by clock name using the `set_scan_path` command. For example:

```
## Sequential cells reg1, reg4 and reg5 will be grouped
## together in single scan chain.
set_scan_group SG1 -include_elements [list reg1 reg4 reg5]
```

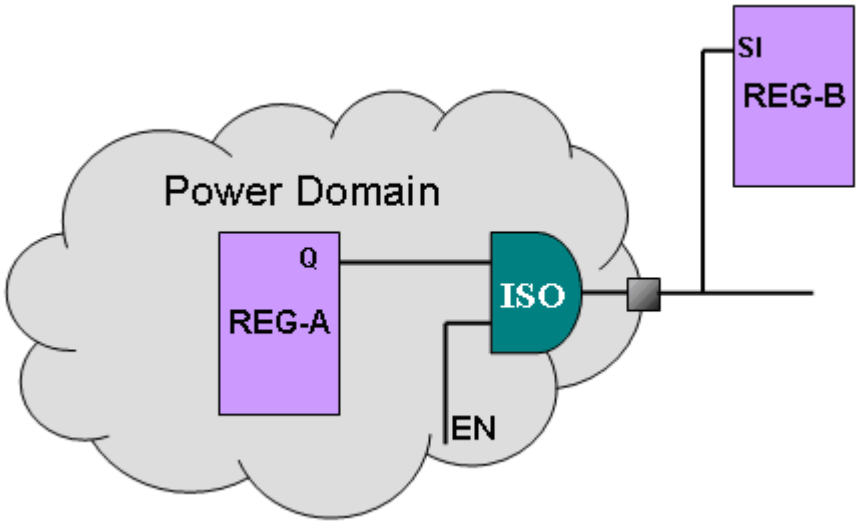
```
## If you would further like to restrict and have only these 3 sequential
## cells in single scan-chains, you can specify the following
set_scan_path -include_elements [list SG1] -complete true
```

If running the `set_scan_path` command results in violating the voltage mixing or power domain mixing specification, the `preview_dft` and `insert_dft` commands issue warning messages about the violations, and then continue.

Scan Insertion

The `insert_dft` command works with power domains, level shifters, and isolation cells. Scan insertion does not remove any existing level shifters and isolation cells. If the special cell is within a block, the `insert_dft` command reuses the existing hierarchical port. This behavior is illustrated in Figure 5-6.

Figure 5-6 Scan Insertion Through Isolation Cell



To prevent the `insert_dft` command from reusing existing multivoltage cells, use the following command:

```
set_scan_configuration -reuse_mv_cells false
```

During the insertion of the scan chains, it is very common for new hierarchical ports to be created in the design to connect the scan chains across the logic hierarchy. These new hierarchical ports are protected by the insertion of level shifter and isolation cells as needed.

The multivoltage functionality for adaptive scan in DFT MAX is the same as that of standard scan. The compressor and decompressor logic is created in the correct voltage corner at the top level. Also, the scan insertion (for both internal scan and scan compression test modes) respect the power domains and special cells. Typically, adaptive scan creates numerous internal scan chains, each crossing the power domain boundary. The large number of level shifter and isolation cells created for these power domain crossings may impact the design area.

The `insert_dft` command automatically inserts level shifter and isolation cells for the newly created test ports. The new hierarchical ports automatically accept the operating condition of the logic hierarchy. The `insert_dft` command inserts the necessary level shifter and isolation cells according to UPF specification.

The `check_mv_design` reports missing level shifters and isolation cells at the power domain interface. For a detailed report, use the `-isolation`, `-level_shifters`, and `-verbose` options in the command.

Retention Registers

If retention registers are used in the design, the states of the retention save and restore signals must be constrained to the powered-up state for DFT implementation. The following commands set these constraints:

```
set_dft_signal -view existing_dft -type Constant \
 -port SAVE -active_state non-save-value
set_dft_signal -view existing_dft -type Constant \
 -port RESTORE -active_state non-restore-value
```

SCANDEF File

The recommended DFT flow is to stitch the scan chains during the logic synthesis stage and then reorder them during the physical implementation stage. All scan synthesis is performed using DFT Compiler inside Design Compiler. After completing scan synthesis, physically aware scan chain reordering is performed in IC Compiler during placement optimization.

IC Compiler reads the DFT-related information that is needed to perform physical reordering through the DEF format (using the SCANDEF file). The SCANDEF file is an ASCII file that specifies a list of stub chains that can be reordered by IC Compiler based on the DFT configuration specified in logic synthesis. Each stub chain defines a scan chain segment whose endpoints are an I/O port, a lockup latch, or a multiplexer.

Therefore, after performing scan insertion in Design Compiler, the scan chain definitions should be saved in a SCANDEF file by using `write_scan_def` command. For example,

```
write_scan_def -out ./DB/dft.scan.def
```

SCANDEF functionality supports scan chains containing level shifters and isolation cells, and it also honors power domains.

For information on DFT and multivoltage designs, see the *DFT Compiler Scan User Guide*.

Reporting

Design Compiler has several reporting capabilities for multivoltage designs. The most useful commands are `report_timing -voltage`, which returns the voltage value for each cell in the selected timing path, and `report_hierarchy`, which provides a summary of the voltage applied to the hierarchical blocks in the design.

Examples of these commands are shown below:

```
report_timing -attritbutes -transition_time -capacitance \
              -voltage -nosplit

report_hierarchy -nosplit -noleaf
```

Writing Out the Design Data

After the logic synthesis is complete, the design can be saved to a .ddc file or a Milkyway CEL. IC Compiler reads the .ddc file or the Milkyway CEL to perform physical implementation. For example,

```
change_names -rules verilog -hierarchy
write -format verilog -hierarchy -output compile.v
write_sdc -nosplit compile.sdc
write_scan_def -output compile.scandef
write_script -output compile.script
write -format ddc -hierarchy -output ./DB/compile.ddc
write_milkyway -output compile
save_upf upf1.upf
```

It is recommended that you use the .ddc format to transfer design information between two Design Compiler sessions. The Milkyway CEL view should be used to transfer the results of logic synthesis to other tools such as IC Compiler. IC Compiler can read both the .ddc format and Milkyway CEL view.

For more information about specific commands, refer to the *Power Compiler User Guide*.

Design Planning Using IC Compiler

IC Compiler can be used for design planning of flat physical designs. This includes basic floorplanning capabilities such as design initialization, power planning, and fast placement of standard cells and macros.

When you start IC Compiler, the UPF mode is enabled by default. To invoke IC Compiler in non-UPF mode, use the `-non_upf_mode` switch when you invoke the tool. You can use the `shell_is_in_upf_mode` command to check the current mode of the IC Compiler shell. This command returns 1 when the shell is in UPF mode and 0 otherwise.

Design planning for low-power designs involves some additional considerations. The major floorplanning features required in multivoltage designs are creating voltage areas, creating multiple supply net routing, inserting multiple-threshold CMOS (MTCMOS) power switch cells, and carrying out supply network analysis.

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.

Always-on synthesis may require specific spots in the floorplan where the supply is always active. Always-on cells can be placed only on those spots that do not require any interface cells.

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.

To create a voltage area, use the `create_voltage_area` command:

```
create_voltage_area
 [modules]
 [-name voltage_area_name]
 [-power_domain power_domain_name]
 [-coordinate llx1 lly1 urx1 ury1]
 [-guard_band_x guard_band_width]
 [-guard_band_y guard_band_width]
 [-color color] [-cycle_color]]
 [-is_fixed]
 [-target_utilization utilization_value]
```

In a UPF flow, when you use the `-power_domain` option, the cells belonging to the specified power domain are automatically assigned to the voltage area. in that case, you do not use the `-name` option and you do not list of modules assigned to the voltage area, as that information comes from the power domain definition.

For example:

```
create_voltage_area -power_domain MULT \
   -coordinate {50.140 50.140 300.00 210.00} \
   -guard_band_x 2 -guard_band_y 2
```

Both rectangular and rectilinear shapes are supported for voltage areas. For an explanation of these command options, see the *IC Compiler Implementation User Guide*.

The specified voltage area is the physical placement area for the logic belonging to the corresponding voltage domain. Except for level shifters, all cells associated with a voltage area operate at the same process, voltage, and temperature (PVT) operating condition values.

If you define voltage areas during floorplanning, the floorplan must contain a default voltage area for the top-level cells and at least one other voltage area. If you directly create the voltage areas by using the `create_voltage_area` command, the tool automatically derives a default voltage area for the top-level leaf cells and level shifters that do not belong to any block.

Supply nets, which are not part of a logic netlist, are required to create power routing. Multivoltage designs typically deal with multiple power and ground nets. This means power routing is always a challenging task for multivoltage designs. The supply net objects, created along with power domains, help in this task by guiding the creation and connection of real supply nets.

Supply net objects can also help identify the power and ground connections to be made between the standard cell power pins inside the power domain and the real supply net. Using this information, the `derive_pg_connection` command makes incremental power connections automatically for any new cells added during optimization. Starting with the

C-2009.06-SP1 release, `derive_pg_connection` is performed implicitly by the `place_opt`, `clock_opt`, and `route_opt` commands. Starting with the C-2009.06 release, `derive_pg_connection` support power and ground connection for physical-only cells based on the UPF definitions.

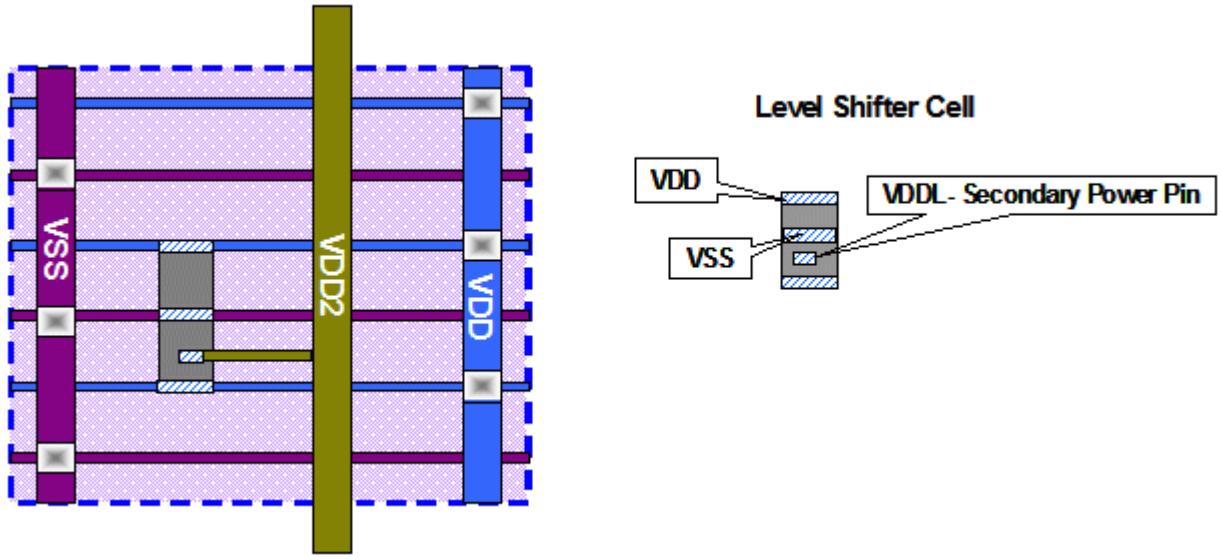
IC Compiler has a set of power routing commands that can be used to create power straps and rings for the voltage areas. Starting from reliability constraints, power network synthesis automatically generates power supply routing for certain design regions, such as voltage areas. In addition, power network analysis provides voltage drop and electromigration information.

Power-Related Cell Placement

Power-related cells such as level shifters, isolation cells, retention registers, always-on cells, and power switches typically have multiple power and ground pins and are taller than standard cells. As a designer, you need to make sure that all the power pins are connected and correctly routed to the power nets.

For example, consider the low-to-high level shifter cell with two power pins shown in Figure 5-7. The cell is taller than a standard cell and has 2X height.

Figure 5-7 Level Shifter With Dual Power Pins

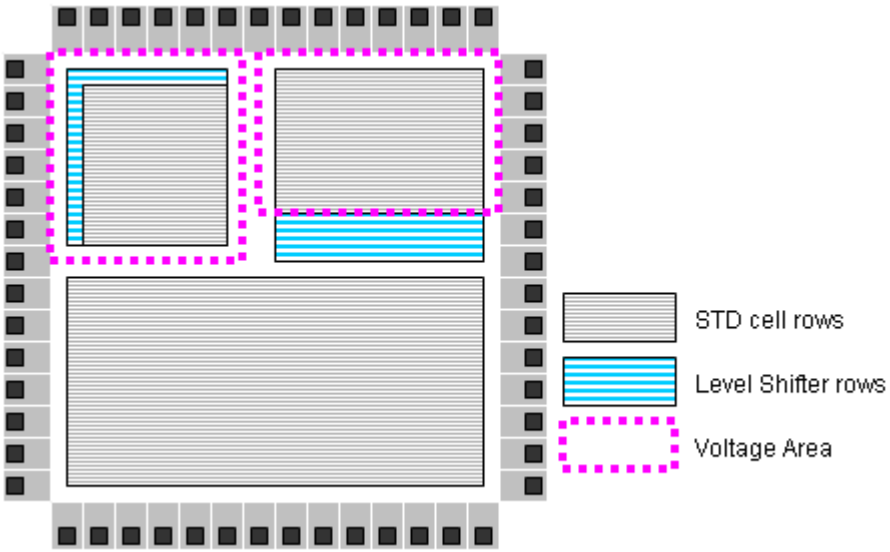


The secondary power pin, VDDL, should be connected to the power supply net VDD2. For power supply routing, VDDL needs to be routed to the VDD2 strap. However, this cannot be achieved with the normal signal or power routing method. A special type of routing, called net mode routing (specified with the command `preroute_standard_cell -mode net`), is required to connect the secondary pin to the nearby power strap.

The layout of the special cell in Figure 5-7 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 5-8 shows an example of what separate level-shifter placement areas might look like.

Figure 5-8 Multivoltage Floorplan



Power switch cells require special placement near the power straps. The `create_power_switch_array` command is used to insert and place the switch cells automatically on the specified insertion grid. After placing the switch cells, the sleep net, which connects all the sleep pins of switch cells, is connected by using the `connect_power_switch` command.

Another aspect of design planning is the selection of the always-on cell strategy. If always-on cells with dual power pins are used for always-on synthesis, these cells can be placed along with standard cells. However, if normal buffers (with a single power supply) are used

in always-on paths, you need to make sure that the cells in the always-on path are pulled into a specific placement area which has constant power supply. This type of special placement area for always-on cells is called an exclusive move bound.

A move bound can be defined as exclusive bound and assigned with the always-on cells, as shown in the following example:

```
## SINGLE POWER RAIL AO STRATEGY
set_always_on_strategy -object {MODULE_A} -cell_type single_power

## CREATE MOVE BOUND FOR AO CELLS
create_bound –name AO_BOUND \
             -exclusive \
             -coordinates {120 140 160 180}

set_power_guide –name AO_BOUND \
                -type always_on
```

For more information about always-on marking and implementation, see the chapter called “Multivoltage and Multisupply Designs and Design Flow” in the *IC Compiler Implementation User Guide*.

Power Planning

After you complete the design planning process and have a complete floorplan, you can perform power planning with the following commands:

- `create_rectangular_rings` (Preroute > Create Rectangular Rings in the GUI) adds power and ground rings to the power network.
- `create_power_straps` (Preroute > Create Power Straps) adds power and ground straps connected to the rings.
- `preroute_standard_cells` (Preroute > Preroute Standard Cells) connects power and ground pins of standard cells to the straps and rings of the power mesh.
- `synthesize_fp_rail` (Preroute > Synthesize Power Network) connects power and ground pins of standard cells to the straps and rings of the power mesh.

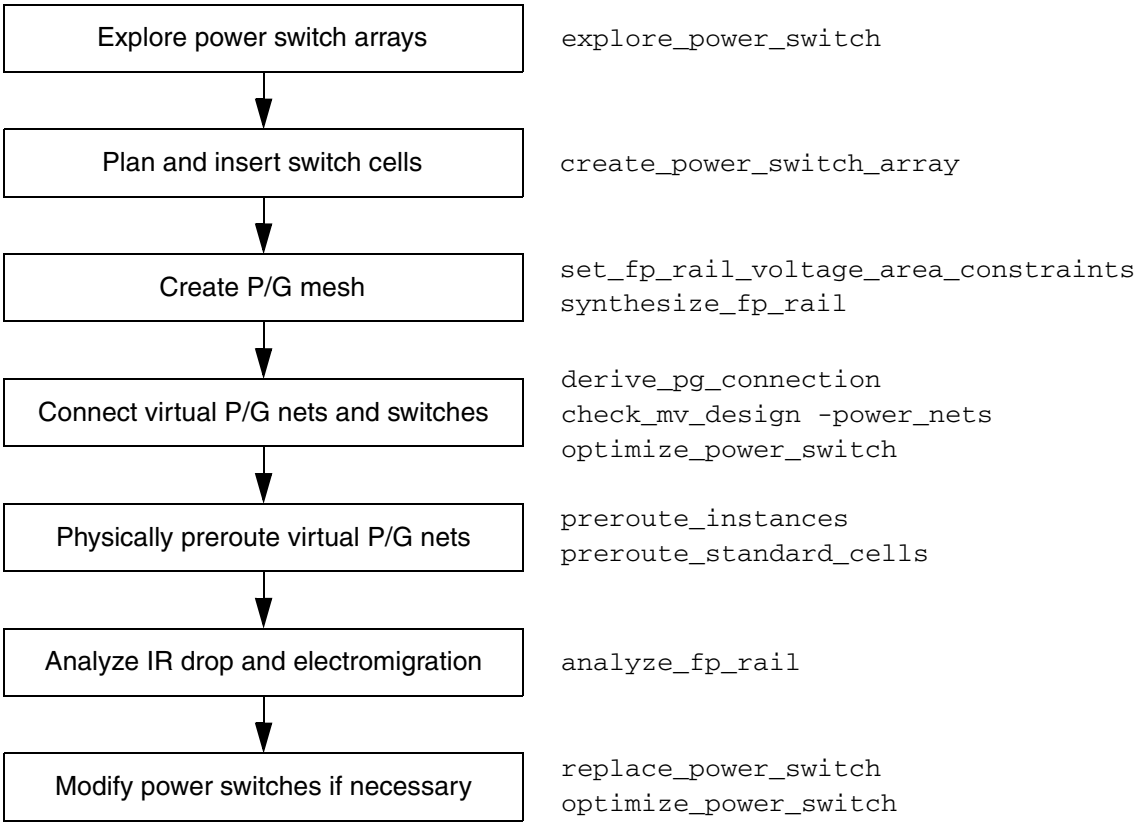
A design with power gating requires a network of power-switching cells distributed physically around or within each power-down block. This network consists of PMOS header switches between VDD and the block power supply pins or NMOS footer switches between VSS and the block ground pins. The following commands support the planning of power switch placement:

- `map_power_switch` (a UPF command) specifies which power switch cell is to be used to implement a specified switch instance in a specified power domain.

- `explore_power_switch` can be used to explore and estimate the placement of power-switching cells in conjunction with a virtual power and ground network.
- `create_power_switch_array` defines a 2-dimensional grid on which the power-switching cells are placed in the floorplan and adds the cells to the logical netlist.
- `connect_virtual_pg_net` creates virtual power and ground nets and connects the power-switching cells to those nets.
- `optimize_power_switch` resolves IR drop problems and optimizes the power-switching cells by automatically sizing the cells larger where required to meet IR drop requirements and sizing them smaller where IR drop requirements are exceeded.
- `connect_power_switch` connects the power-switching control signals to the switch inputs of the power-switching cells.

Figure 5-9 shows the typical flow for planning power switches.

Figure 5-9 Power Switch Planning Flow



The `analyze_fp_rail` command performs power network (IR drop) analysis for both real and virtual power nets to the power-switching cells, for complete or incomplete power nets. The `synthesize_fp_rail` performs power network (IR drop) analysis for both real and virtual power nets to the power-switching cells, for complete or incomplete power nets.

The `set_fp_rail_strategy` command sets various parameters for power network synthesis and power network analysis such as macro connection and blocking, strap alignment, chimney checking, power pad checking, and completeness checking. The `set_fp_rail_constraints` command sets the constraints for power network synthesis, including power strap layer constraints, power ring constraints, and global constraints. The `set_fp_rail_voltage_area_constraints` command sets the power network synthesis constraints for a specified voltage area.

The `derive_pg_connection` command performs automatic power/ground connection for power/ground pins of cells of the design. The `-reconnect` option causes any existing power/ground connections to be replaced with new ones. The `connect_pg_nets` command can be used to connect the power supply nets to physical-only cells.

The `commit_fp_rail` commits the power network, including power and ground wires and vias, based on the results of power network synthesis.

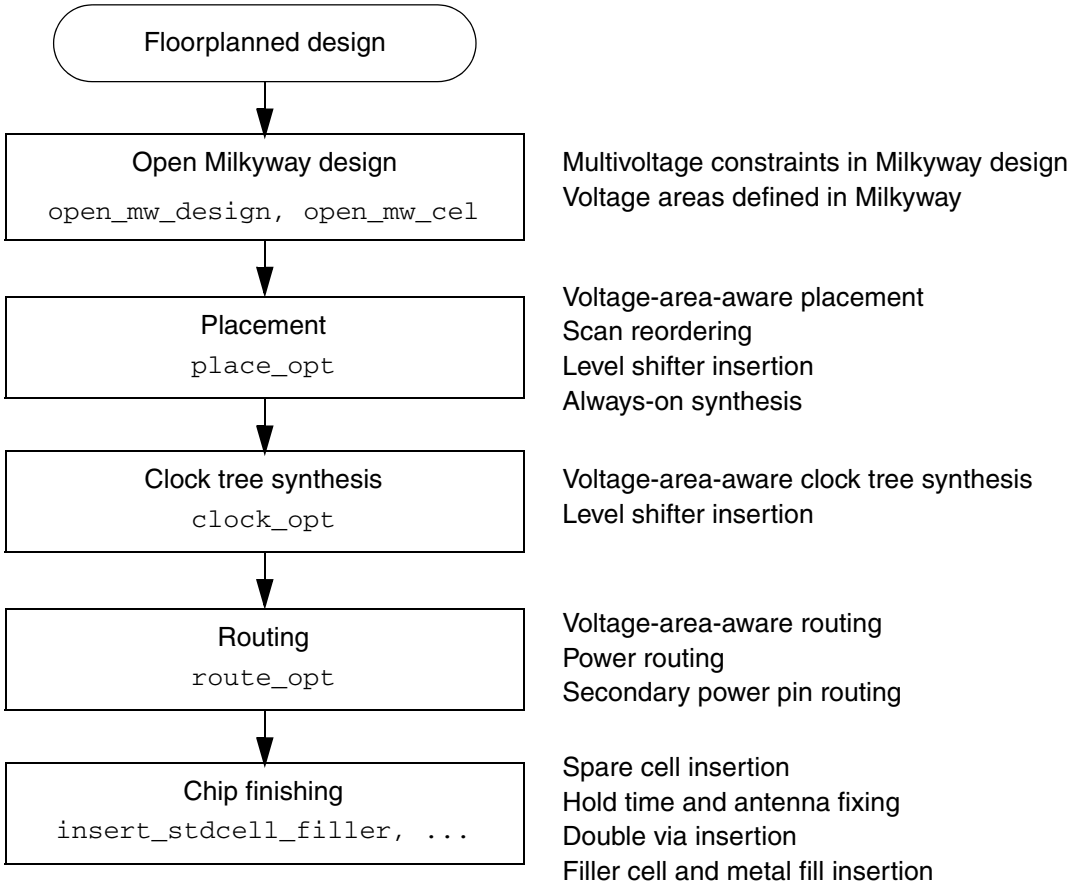
For more information about power network analysis and power network synthesis in design planning, see the *IC Compiler Design Planning User Guide*.

Physical Implementation Using IC Compiler

After the floorplan is completed with the rows, power planning, and I/O placement determined, physical synthesis and implementation can begin. IC Compiler provides a broad set of features to perform a physically flat implementation of the design.

Figure 5-11 shows the general steps in the physical implementation flow and the low-power support features of the flow. Starting with a design previously floorplanned in IC Compiler, you read in the design from the Milkyway database. The design already has the multivoltage constraints and voltage areas previously defined in Design Compiler and IC Compiler. You then perform voltage-area aware placement, clock tree synthesis, and routing. Finally, you perform the chip finishing steps.

Figure 5-10 Low-Power Physical Implementation Flow



In IC Compiler, use `set_operating_condition` for the top level and `set_voltage` for all power supplies of the design:

```
icc_shell> set_operating_condition opcond_name

icc_shell> set_voltage value -object_list { supply_nets }
```

Before performing optimization, perform a multivoltage design check with the `check_mv_design -verbose` command. This command checks the multivoltage constraints, isolation requirements, and connection rules. It generates detailed reports on any violations.

The core commands `place_opt`, `clock_opt`, and `route_opt` can optimize timing by adding new instances to the correct logic hierarchy, and can legally place standard cells, level shifters, and isolation cells while honoring the voltage areas.

IC Compiler automatically preserves any existing level shifters, isolation cells, and enable level shifters that are already part of the design. If required, level shifters cells can also be incrementally inserted with the `insert_level_shifters` command. Enable level shifters and isolation cells can be inserted with the `insert_isolation_cell` command. However, the introduction of level shifters or isolation cells should occur as early as possible in the design flow to help maintain control of critical timing paths and avoid timing problems at the implementation stage.

The ability to recognize the cells at the power domain boundary is the key to appropriately setting the dont_touch attributes. IC Compiler recognizes the level shifters, isolation cells, and enable level shifters automatically and propagates the dont_touch attributes to the nets that must preserve the voltage consistency of the design.

IC Compiler places level shifters, isolation cells, and enable level shifters close to the voltage area boundaries. This feature reduces the routing lengths crossing the voltage areas because IC Compiler sets dont_touch attributes on those wires.

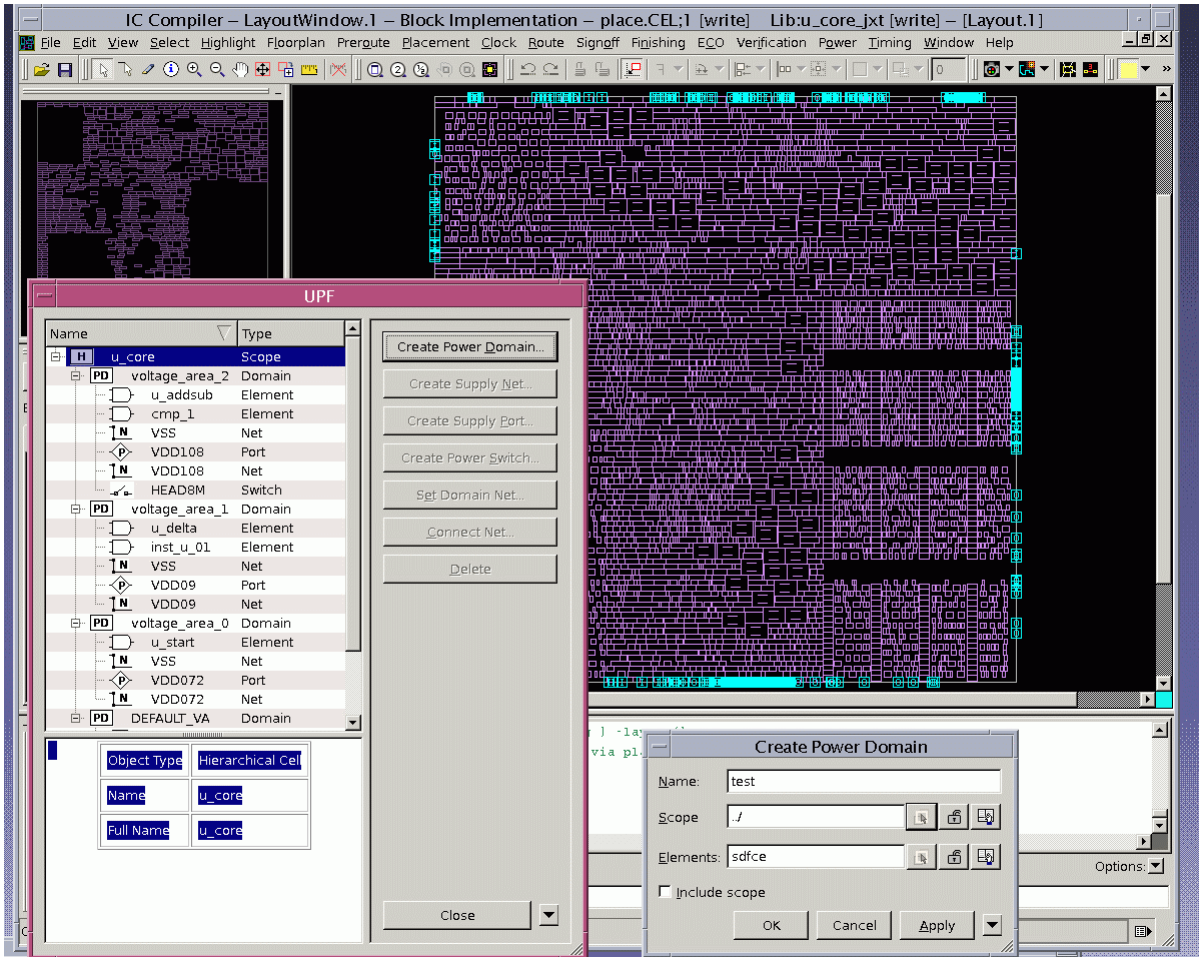
The power domain and voltage area definitions can be reported at any time with the `report_power_domain` and `report_voltage_area` commands. If new power domains or voltage areas need to be defined in IC Compiler, they can be created with the `create_power_domain` and `create_voltage_area` commands.

The following commands are often used for checking and reporting multivoltage designs. For detailed information on any command, see its man page.

```
check_design
check_isolation_cells
check_mv_design
report_cell
report_delay_calculation
report_hierarchy
report_isolation_cell
report_level_shifter
report_operating_conditions
report_power_domain
report_power_switch
report_retention_cell
report_supply_net
report_supply_port
report_target_library_subset
report_timing
report_voltage_area
```

The IC Compiler graphical user interface (GUI) provides a means to view and edit the specified power supply structure. Use Power > UPF to open the UPF dialog box. In the dialog box, you can view the current UPF configuration and make changes such as creating power domains and connecting supply ports. See Figure 5-11.

Figure 5-11 UPF Dialog Box in IC Compiler



Placement and Optimization

When the `place_opt` command performs placement and physical synthesis, it takes into account the voltage areas, power domains, instance-based targeting, always-on logic, and special cells protecting the power domain interfaces (isolation cells, level shifters, and enable level shifters). All the optimization commands are voltage-area-aware. This means that each design instance is assigned to a single voltage area, the voltage area is exclusively allocated to a certain logic hierarchy partition, and any new instance created for a certain logic hierarchy is legally placed within the voltage area boundary and subjected to the same operating conditions.

In addition, automatic high-fanout net synthesis (part of the `place_opt` command) honors voltage areas. Buffer trees are built with dedicated subtrees for the fanout cones in each voltage area. In particular, automatic high fanout synthesis selects buffers according to the associated operating condition. After running the `place_opt` command, it is good practice to check whether new level shifters are needed, because new hierarchical ports may have been created.

IC Compiler minimizes the formation of nets that cross multiple voltage area boundaries. When buffering a net that crosses a voltage area boundary, IC Compiler divides the net into segments, with each segment confined to a single voltage area, and restricts buffer insertion to these segments. It also ensures that a tie cell in one voltage area does not feed a cell in another voltage area.

IC Compiler respects the logic hierarchy when handling the tie-high and tie-low cells. It inserts tie cells by default as long as a dont_touch or dont_use attribute is not in the library.

Placement recognizes the cells working at the power domain interface and places those cells near the related voltage area boundary. Moreover, IC Compiler routing estimation is capable of detouring around voltage areas. Therefore, the `place_opt` command can correctly optimize the design logic within a voltage area as well as the logic outside the area.

As noted earlier, level shifters can be taller than the standard cells. If the height of the level shifter is double or triple the height of a standard cell, then it is recommended to have only one type of row where all standard cells and level shifters can be placed together. However, the possibility of placing level shifters side by side with standard cells also depends on how the level shifter has been laid out. Due to DRC rules, in some cases level shifters may require special placement attention.

All the core commands in IC Compiler perform always-on synthesis for the selected always-on paths. The methodology is same as in Design Compiler. In addition, if the single-power-rail strategy is used with an always-on exclusive bound, the optimizations make sure that the newly added always-on cells are automatically pulled into the always-on bound.

When multicorner-multimode technology is used, optimization works on the worst timing violations across all scenarios, thus eliminating the convergence problems observed in sequential approaches. Optimization can be performed for DRC, setup and hold, or setup only.

Scan Chain Reordering

Another step performed in the physical synthesis stage is scan chain reordering, which can reduce the scan wire lengths and congestion and to improve routability. To carry out scan reordering, IC Compiler needs to read in the SCANDEF file generated by Design Compiler,

which contains the scan chain definitions. After that, the `place_opt` and `clock_opt` commands can reorder the scan chains based on the physical locations of the registers. For example,

```
## READ SCANDEF
read_def -verbose ./dft.scan.def
..
## RUN PLACE_OPT WITH SCAN REORDER
place_opt -optimize_dft
...
## RUN CLOCK_OPT WITH SCAN REORDER
clock_opt -optimize_dft
```

The SCANDEF file defines groups of scan chains called partitions. Each partition contains one or more scan chains that are allowed to exchange flip-flops. A partition can have only one operating condition and one power domain.

You can optionally use the `set_optimize_dft_options` command to set the parameters for DFT optimization prior to using the `place_opt` command. To perform scan chain optimization as a separate process, use the `optimize_dft` command.

Level Shifters

Level shifters should be created in the netlist as early as possible, in the synthesis stage using Design Compiler. In Design Compiler, the `compile_ultra` or `compile` command automatically inserts level shifters where required.

In some cases, IC Compiler must add level shifters on new paths that cross voltage area boundaries, for example, those resulting from scan chain insertion, test port hookup, or logical feedthrough insertion. You can insert level shifters with the `insert_level_shifters` command. Before you do so, be sure that the voltages have been set on the design with the `set_voltage` command, the level shifter libraries have been specified in the `target_library` variable, and the level shifter strategy and threshold have been defined.

A typical level insertion flow consists of commands similar to the following:

```
set_level_shifter_threshold -voltage 0.1
set_level_shifter_strategy -rule all
check_mv_design -level_shifters
insert_level_shifters
```

The `check_mv_design -level_shifters` command checks and reports level shifter violations involving nets, cells, and operating conditions. You can run this command at any point in the flow.

To remove level shifters that have been inserted by IC Compiler, use the `remove_level_shifters` command.

Isolation

An isolation cell selectively drives a known signal value at the edge of a shutdown voltage area. An enable-type level shifter performs both level shifting and isolation functions. During logic synthesis, Design Compiler can replace isolation cells with enable level shifters if the appropriate cells are available in the libraries.

Isolation cells can be instantiated at the RTL level, or they can be manually inserted by using the `insert_isolation_cell` command throughout the flow. Enable level shifters cannot be inserted by a user command, but they can be inserted at the RTL level or inserted automatically during compilation to replace existing isolation cells.

There are several ways that isolation cells and enable-type level shifters might be connected to the input and output nets of the voltage areas. It is possible to unintentionally introduce redundant isolation cells into the RTL description or to fail to specify necessary isolation cells. You can use the `check_isolation_cells` command to check for these conditions.

Always-On Synthesis

An always-on path is a control net that is driven from an always-on power domain and ends within a shutdown power domain, and which must remain on during shutdown. Some examples of always-on signals include the control pins of power switch cells, isolation cell enable signals, and feedthrough nets.

IC Compiler automatically recognizes the always-on characteristic of enable pins of special-purpose, dual-power cells and the input pins of single-power standard cells placed in always-on power wells. This includes isolation cells, enable level shifters, power switch cells, and retention registers. Other types of always-on signals might need to be manually marked. IC Compiler determines the always-on paths by tracing back from the always-on pins along the fanin logic cone of each pin.

During execution of the `place_opt` command, IC Compiler performs optimization of always-on paths, including buffering and cell sizing. It uses ordinary cells for the portion of an always-on path outside of the shutdown power domain, and it uses always-on cells for the portion within the shutdown power domain.

IC Compiler performs always-on synthesis only when the target library contains an always-on buffer and an always-on inverter. The target library can have special-purpose cells marked for use in always-on paths. Alternatively, you can identify library cells or pins in your IC Compiler session by setting the library cell or library pin attributes. For example, you can use the following syntax:

```
set_attribute [get_lib_pins lib_name/cell_name/pin_name]  always_on true
```

IC Compiler can use either single-power or dual-power cells for always-on synthesis. Placement of standard, single-power cells is restricted to the site rows of special-purpose, always-on power wells created within the shutdown block's voltage area. Dual-power cells can be placed anywhere in the shutdown power domain, but the selected locations must allow for the routing of secondary power lines.

To use single-power cells, you need to set the always-on strategy to single-power mode, create an exclusive move bound to contain the new cells, and define the move bound as a always-on power guide. Here is a typical script to perform these steps:

```
set_always_on_strategy -object_list PD1 \
  -cell_type single_power
create_bounds -name AON_RFF_BUFS \
  -coordinate {547 131 573 893} \
  -exclusive [list [get_always_on_logic]]
set_power_guide -name AON_RFF_BUFS -type ao
```

To get information about power guides that have been defined, use the `report_power_guide` command. To remove a move bound as a power guide, use the `unset_power_guide` command.

The `check_mv_design -connection_rules` command reports always-on problems such as a normal cell driving an always-on net in a power-down domain, an always-on cell driving a normal net, and certain types of pass gate connection problems.

If an always-on net passes through a shutdown voltage area and needs to be buffered, always-on buffers must be used in the shutdown area. To allow IC Compiler to generate these types of feedthroughs, use a flow similar to the following command sequence:

```
set_fp_voltage_area_constraints -allow_feedthroughs true
...
#  Global routing here
...
analyze_fp_routing -output_feedthroughs nets
set_fp_voltage_area_constraints -allow_feedthroughs true \
  -exclude_feedthroughs [get_nets nets_to_exclude]
analyze_fp_routing -finalize_pins_feedthroughs voltage_areas
set_attribute [get_pins feedthrough_output_port] always_on true
...
place_opt
...
clock_opt
...
route_opt
```

Clock Tree Synthesis

The `clock_opt` command is multivoltage-aware. No special setup is required. The clock tree synthesis engine in IC Compiler recognizes the logic hierarchy associated with the voltage area and creates the clock tree bottom-up by clustering sink points from the same voltage area. After the clock subtrees are built for each voltage area, clock tree synthesis joins the subtrees at the root of the clock net.

Here is an example of a clock tree synthesis script:

```
## SET CTS OPTIONS
set_clock_tree_options -max_transition 0.5 -max_capacitance 0.6

## CLOCK TREE REFERENCES
set_clock_tree_references -references { BUF2 BUF4 }

remove_clock_uncertainty [all_clocks]

## CLOCK_OPT
clock_opt -fix_hold_all_clocks
```

For bottom-up clock tree synthesis, IC Compiler performs endpoint clustering based on voltage areas. It builds a separate clock subtree for each voltage area, without crossovers between these subtrees.

You should insert level shifters and isolation cells as needed on clock nets that cross power domains before running clock tree synthesis. To check for proper insertion, use the `check_mv_design -level_shifters` and `check_mv_design -isolation` commands.

The `clock_opt` command supports multicorner-multimode technology. Clock tree synthesis is performed only in the clock scenario and optimizations are performed for all scenarios.

Routing

By default, IC Compiler observes the following rules while routing multivoltage designs:

- Nets connecting within a single voltage area are restricted to that voltage area.
- Nets connecting cells outside a voltage area must detour around that voltage area.
- Net that cross voltage area boundaries must be minimized.

Feedthroughs that cross voltage areas are allowed if you execute the following command:

```
set_fp_voltage_area_constraints -allow_feedthroughs true
```

To perform power and ground routing to standard cells, use the `preroute_standard_cells` command. This command connects power and ground pins in the standard cells to the power and ground rings or straps, and it connects power and ground rails in the standard cells.

The `preroute_standard_cells` command has a `-mode` option, which lets you select one of three power/ground connection modes: `rail`, `tie`, or `net`:

- Use `preroute_standard_cells -mode rail` to connect standard cells by rails.
- Use `preroute_standard_cells -mode tie` to connect standard cells by tying all the power and ground pins to nearby targets.
- Use `preroute_standard_cells -mode net` to connect the pins of standard cells to a nearby power ring or strap; additional command options let you specify the layers, widths, and maximum allowed fanout for the connections. Each routing cluster has a limited number of secondary power pins connected to the nearest power strap.

The net mode is useful for routing the secondary power pins of dual-power cells. You can use the `-nets` option to restrict the routing to specified power and ground nets.

Multicorner-Multimode Technology

Multicorner-multimode technology is a key capability required to optimize “dynamic voltage and frequency scaling” types of multivoltage designs. Using multicorner-multimode technology, IC compiler can optimize and analyze the design concurrently for all its operating voltage and frequency states.

In IC Compiler, a design scenario refers to a functional mode, a corner, or a combination of both functional mode and corner. With concurrent optimization and analysis, all scenarios are handled simultaneously for timing, area, design rule checking, signal integrity, and power.

A multivoltage design that exhibits multiple voltage states can be treated as a multicorner design (one corner for each set of operating voltage states). Scenarios are created by using the `create_scenario` command. A scenario definition must be followed by setting the scenario-specific constraints: operating conditions, Synopsys Design Constraints (SDC), and TLUPlus models. For example,

```
## CREATE SCENARIO S0
create_scenario S0
source ./cstr_S0.sdc
set_operating_conditions -max WCCOM -min BCCOM
set_voltage 1.08 -object_list {VDD VDDM VDDMS VDDI VDDIS VDDG VDDGS}

## CREATE SCENARIO S0
create_scenario S1
```

```
source ./cstr_S1.sdc
set_operating_conditions  -max WCCOM -min BCCOM
set_voltage 1.08 -object_list {VDD VDDM VDDMS}
set_voltage 0.865 -object_list {VDDI VDDIS VDDG VDDGS}
set_voltage 0 -object_list {VSS}
```

The `place_opt` and `route_opt` commands are fully supported in multicorner-multimode flows. As for the `clock_opt` command, clock tree synthesis occurs only in the active scenario. However, all subsequent clock optimizations are performed across all scenarios.

For multivoltage designs, it is important to protect the nets connecting level shifter and isolation cells to the power domain boundary in all scenarios. In this case, the tool preserves the dont_touch attribute on these nets across all scenarios. You can run the `check_mv_design` command on a per-scenario basis and then insert level shifters or isolation cells if required.

Power Optimization

During physical synthesis, the `place_opt` command can concurrently optimize the design for dynamic and leakage power. The `set_power_options` command has different options to select the type of power optimization and effort level to be used by the `place_opt` command. For details on the power optimization options, refer to the *IC Compiler Implementation User Guide*.

A very important application of instance-based targeting is leakage optimization. Instance-based targeting can be applied selectively on timing critical blocks in the design by specifying multiple-threshold libraries as target libraries for those blocks. For example,

```
# IBT: SET MULTI-VT LIBRARIES ONLY FOR Multiplier BLOCK
set_target_library_subset "lib_hvtwc.db" -top
set_target_library_subset "lib_hvtwc.db lib_lvtwc.db" -obj Multiplier

# SET POWER OPTIONS
set power_keep_tns true
set_max_leakage_power 0.0 mW
set_power_options -leakage true

# RUN PLACE_OPT
set adaptive_leakage_opto true
place_opt -power
```

Timing and Signal Integrity

The IC Compiler timing engine is multivoltage-aware and signal-integrity-aware. It can detect those signals that connect the driver and load across power domains that operate at different voltage levels. Where level shifters are not inserted, the timing engine scales the transition time according to the voltage levels of the domains involved.

Delay calculation and transition scaling details can be obtained by using the `report_delay_calculation` command. For example,

```
Rise Delay Voltage Adjustment
   Driver voltage: 1.08
   Load voltage: 1.32
   Driver delay trippoint: 50
   Load delay trippoint: 50
   Driver transition time: 1.25402
   Driver upper trippoint: 90
   Driver lower trippoint: 10

Adjustment = driver_transition_time * (load_voltage *
load_delay_trippoint -driver_voltage * driver_delay_trippoint) /
(driver_voltage * ( upper - lower ))
    = 1.25402 *
(1.32 * 50 - 1.08 * 50) / (1.08 * ( 90 - 10 )) = 0.174169
Adding 0.174169 to original rise delay (0) gives 0.174169
```

The `report_timing` command has an option, `-voltage`, that shows the operating conditions used by the timing engine for each instance in the timing report.

During multicorner-multimode analysis, timing analysis is carried out on all scenarios concurrently, and costing is measured across all scenarios. As a result, the timing and constraint reports show worst-case timing across all scenarios.

The timing engines in IC Compiler and PrimeTime take into account the voltage impact during crosstalk-aware timing analysis. The timer individually scales voltage levels based on the delay contributions of aggressors and victims. Signal integrity analysis and optimization are supported in both multicorner and multimode flows.

Saving the Design in the Milkyway Database

When the physical implementation is complete, the design can be saved into a CEL view. By default, the `save_mw_cel` command saves all the scenarios into the CEL view.

The Milkyway CEL can be read into StarRC to generate parasitic information in SPEF format. The Verilog netlist and Synopsys Design Constraints (SDC) file should be dumped out for sign-off static timing analysis in PrimeTime. For example:

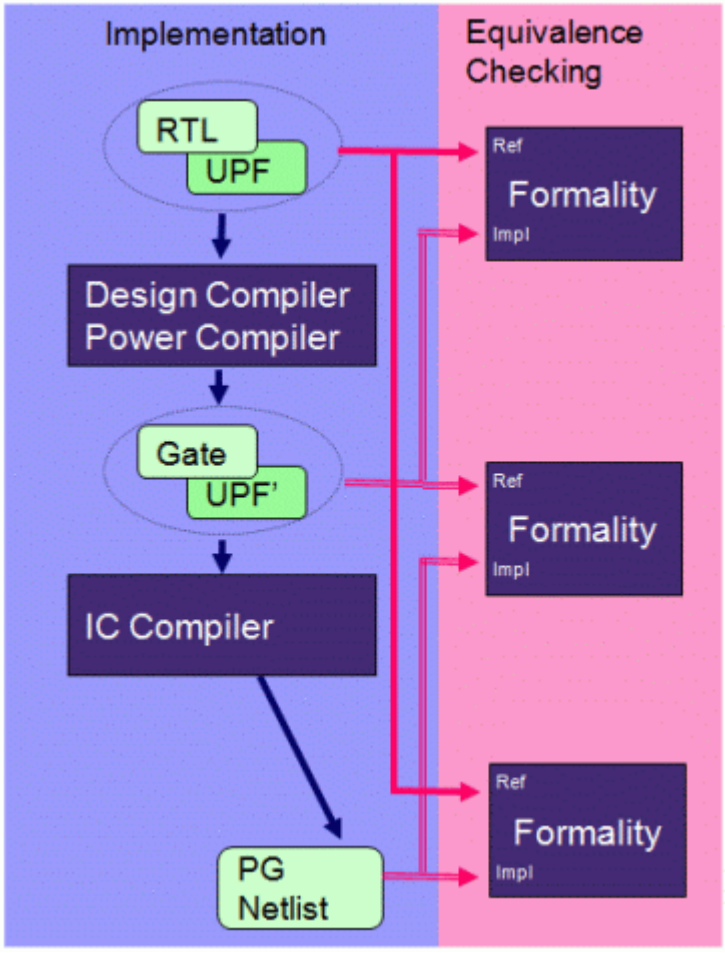
```
change_names -rules verilog -hierarchy

write_script -nosplit -output ./DB/route.script
write -format verilog -hierarchy -output ./DB/route.v
write_link -nosplit -output ./DB/route.link

current_scenario S0
write_sdc -nosplit ./DB/route_S0.sdc

current_scenario S1
write_sdc -nosplit ./DB/route_S1.sdc

save_mw_cel mydesign
```

Formal Verification Using Formality

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

Formality supports verification of low-power design data with UPF as shown in Figure 5-12.

Figure 5-12 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.

Formal verification requires certain preparation steps with respect to libraries and the treatment of level shifters and isolation cells.To run Formality, all related technology libraries must be read in, meaning all logic libraries, including level shifter libraries. Power-aware library cells must have power pins and power-down functions defined. Prior to reading the reference and implementation designs, SVF (Formality guide) files created in Design Compiler must be read by using the `set_svf` command. The SVF file contains information used in verification to improve performance and verification success.

Formality does not accept individual UPF commands entered interactively. The UPF description must be read into Formality with the `load_upf` command. You can use a single, top-level `load_upf` command, and for a hierarchical implementation flow, the top-level UPF script can contain additional `load_upf` commands.

The following script reads in RTL + UPF and a synthesized netlist + UPF' generated by Design Compiler:

```
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 use `load_upf`, check the warnings for conditions such as unread supply pins, possibly indicating a problem with the library cell. For example,

```
Warning:  load_upf: connecting supply net VSS to unread supply pin
  Top/mid_20__UPF_LS/VSSC (LIBNAME/CELLNAME#PWR).
```

Formality modifies the target reference and implementation designs to meet the specifications implied by the respective UPF command files. When a reference block is powered up, Formality verifies its functionality as usual. However, when a reference block is powered down, the compare points of the block are considered don't care. Formality does not allow a reference X originating in an RTL+UPF "off" power domain to control compare points in an "on" power domain. Failing ports are reported if an X leaks out of a power domain in the implementation to downstream compare points, thus detecting the absence or malfunctioning of a needed isolation cell.

By default, Formality accepts and uses the Verilog netlist constructs $isolate, $power, and $retain. To ignore them instead, use the following command in Formality:

```
set hdlin_enable_rtl_power_constructs false
```

Formality recognizes and uses loaded UPF commands that create power domains, supply nets, supply ports, power switches, isolation cells, and retention registers. Because level shifters have no functional simulation effects, Formality does not insert level shifters that perform only buffering. Formality recognizes power state tables defined with the `create_pst` command.

Design Data Modification With UPF

Formality modifies the target design data (reference or implementation) to meet the specification implied by the UPF commands. In each type of comparison, Formality verifies the power-up functionality and reports failing points if the implementation netlist (or netlist plus UPF) powers down in a manner inconsistent with the reference design plus UPF. During verification, when an area in the reference design is powered down, Formality treats the compare points in this area as don't care.

In Formality, the `load_upf` command reads the file containing the UPF commands associated with the design data. It modifies the target design data to meet the specification implied by the UPF commands. As the RTL has been simulated and synthesized using a specific set of UPF commands, Formality must use those same commands to verify the RTL against the netlist. Therefore, UPF commands cannot be issued interactively in Formality. The `load_upf` command must be used after the `set_top` command after the design in the container has been successfully elaborated.

The following example shows part of a script using the `load_upf` command for the reference design:

```
read_verilog -r { top.v block1.v block2.v block3.v block1a.v }
set_top r:/WORK/top
load_upf -r top.upf
```

The `load_upf` command modifies the design data by adding nets and ports as implied by the `create_supply_net`, `create_supply_port`, and `connect_supply_net` UPF commands. It adds ports between hierarchical levels (performs “port punching”) as needed to make the supply connections. It also adds logic to the design based on the `create_power_switch`, `set_isolation`/`set_isolation_control`, and `set_retention`/`set_retention_control` UPF commands. After you execute `load_upf`, the design is modified and you can continue with the rest of the verification run.

Retention Registers

To allow formal verification of netlists synthesized from RTL, the behavior of the netlist must be consistent with the behavior of the RTL simulation. For any condition under which the RTL simulation value is not X for a register next state, primary output port, black-box input pin, or other compare point, the value of the matching netlist compare point must be identical.

In the case of the UPF `set_retention` constructs, this level of consistency is not always possible for the following reasons:

- Design Compiler chooses retention register implementations based on directives (pragmas or commands) outside of UPF.
- 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, do the following:

- In the `set_retention_control` command, for the `-save_signal` and `-restore_signal` settings, use only `high` and `low` options. (The `posedge` and `negedge` settings of the UPF 1.0 standard are not supported by Synopsys tools in the current release.)
- Direct Design Compiler to implement only “clock-free” retention registers.

Static Timing Analysis Using PrimeTime

PrimeTime reads a gate-level netlist from Design Compiler or IC Compiler 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.

The following is a typical sequence of commands used in a PrimeTime timing analysis with UPF-specified power intent.

```
# Read libraries, designs
 ...
read_lib l1.db
read_verilog d1.v
 ...
link_design new_cpu

# Read UPF file

 load_upf my_file.upf

# Define scaling library group
 ...
define_scaling_lib_group {lib_0.9V_0C.db lib_1.1V_0C.db lib_1.3V_0C.db}
...
set_voltage -cell ... -pg_pin_name ... value
# (sets voltage on supply nets or IR drop on power pins)
set_temperature -object_list cell_list value

# Read SDC and other timing assertions
read_sdc d1.tcl

# Read parasitics
read_parasitics my_rc.spef

# Perform timing, signal integrity analysis
report_timing
```

For power analysis using PrimeTime PX, the following additional commands are typically used in the flow:

```
# Perform power analysis
# The most common flow alternatives are outlined below:

# Time-based switching activity file
read_vcd file.vcd

# Control signal state probablilities
read_saif file.saif

# Set switching activity on signal nets
set_switching_activity ...

# Set static values on control signal nets of the power switch
set_case_analysis ...

report_power
```

Voltage Scaling

The multivoltage flow can use either CCS timing or NLDM libraries. A set of CCS timing libraries that cover the range of voltages can be used with the voltage and temperature scaling capability in PrimeTime. Using NLDM libraries requires characterization at each of the voltages used in the design, and you set the `link_path_per_instance` variable to a list, with each list element consisting of a list of instances and the corresponding link paths that override the default link path for each of those instances.

With CCS timing libraries, PrimeTime supports voltage and temperature scaling by interpolating between data in separate libraries that have been characterized at different nominal voltage and temperature values. The delay (CCS timing driver model and receiver model) and timing constraints are scaled. In addition, scaling occurs if there is a mixture of CCS and NLDM data. Scaling between the libraries is done during runtime of the tool.

You can invoke voltage and temperature scaling by using the `define_scaling_lib_group` command. This command specifies the scaling relationships between libraries that have been characterized at different voltages and temperatures and invokes both delay and constraint scaling, as shown in the following example:

```
pt_shell> define_scaling_lib_group \
          {lib_0.9V_0C.db lib_1.1V_0C.db lib_1.3V_0C.db}
```

This command should be issued after the design has been read in. If the design is not already linked, the `define_scaling_lib_group` command automatically links the design and creates scaling relationships between the libraries in each group. If the design is already linked, this command creates scaling relationships without an additional link. You can define multiple scaling groups to cover different portions of your design. However, each scaling group should contain the correct libraries for the type of scaling being performed (voltage, temperature, or voltage and temperature), and each library can be part of only one scaling group.

For more information about voltage and temperature scaling, see the section called “Scaling With CCS Timing Libraries” in the *PrimeTime Advanced Timing Analysis User Guide*.

Setting Supply Voltages and Temperature

The `set_voltage` command specifies the voltage value on a supply net or PG pin of a cell, in volts. You can specify different minimum and maximum delay voltage values. You can use the `-dynamic` and `-min_dynamic` options to specify the dynamic portion of the supply voltage (the portion that can vary across successive clock cycles). The `set_temperature` command specifies the operating temperature for a list of cells, in degrees Celsius. You can specify different minimum and maximum delay temperature values.

PrimeTime determines the cell rail voltage from the following sources of information, in order of increasing priority:

- `voltage_map`
- `set_voltage` on a power net
- `set_voltage` on a power pin
- design operating condition

The `voltage_map` statement in the library description of the cell (in Liberty format) defines the power supplies and default voltage values. The `related_power_pin`, `input_signal_level`, and `output_signal_level` attributes in the library cell description specify which power supply is connected to the pin's transistor stage. The voltage signal level margins are defined by the `input_voltage` and `output_voltage` attributes in the library description.

On-Chip Variation Analysis

In PrimeTime, the operating condition setting does not constrain the design linker. Note that the link is implemented before any SDC input. You can set any operating condition after the linking is complete. If CCS timing scaling library groups are used, the delay calculation is performed at the specified operating condition, as long as it is within the voltage and temperature range of the scaling group. If NLDM libraries are used, it is recommended to use `link_path_per_instance` to assign the appropriate library to an instance. Operating conditions beyond the range of the scaling library group should be avoided. If scaling library groups are not being used, it is recommended to use a library characterized at the desired operating condition for both NLDM and CCS libraries.

On-chip variation analysis is recommended for timing sign-off. Best-case/worst-case analysis can produce results that are too optimistic in certain cases. In multivoltage designs, on-chip variation timing analysis can be performed one corner at a time using derate timing factors to model a slightly better or slightly worse condition around the main corner condition. To run this analysis, you must define a specific set of timing constraints to be targeted on only one corner.

When you use on-chip variation analysis, `set_voltage -max` *`value`* is used for maximum analysis and `set_voltage -min` *`value`* 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. For example,

```
pt_shell> report_delay_calculation -from I1/Z -to I2/B
  From pin: I1/Z
  To pin:   I2/B
  Main Library Units:  1ns  0.001pF  1000kOhm

  arc sense: unate
  arc type:  net

   RC network on pin 'I1/Z' :
   ----------------------------------------------------
   Number of elements = 2 Capacitances + 1 Resistances
   Total capacitance  = 0.815687 pF
   Total capacitance  = 815.686687 (in library unit)
   Total resistance   = 8.323139 Kohm

   Scaling library pin group used for rise and fall.
   Scaling libraries used for receiver model : tc300c_0.85  tc300c_1.05

                                Rise        Fall
   ---------------------------------------------------------------------
   Net delay                 = 2.406542    2.836394  (in library unit)
   Transition time           = 5.312853    4.983530  (in library unit)
   From_pin transition time = 0.804747    0.328441  (in library unit)
   To_pin transition time    = 5.312853    4.983530  (in library unit)
   Net slew degradation      = 4.508106    4.655089  (in library unit)
...
```

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

```
pt_shell> report_power_pin_info [get_cells -hierarchical]
...
Cell             Power Pin Name  Type           Voltage MaxD MinD  Power Net Connected
-------------------------------------------------------------------------------------
U61              VDDC            primary_power     1.1000  1.1000  VDD
U61              VSSC            primary_ground    0.0000  0.0000  GND_main
do_reg_reg_5_    VDDC            primary_power     1.1000  1.1000  VDD
do_reg_reg_5_    VSSC            primary_ground    0.0000  0.0000  GND_main
...
```

To verify the signal-level consistency on the whole design, use the `check_timing` command. In the following example, the `check_timing` report shows some signal-level violations.

```
pt_shell> check_timing -include signal_level -verbose
```

```
Information: Checking 'unconstrained_endpoints'.
Information: Checking 'unexpandable_clocks'.
Information: Checking 'generic'.
Information: Checking 'latch_fanout'.
Information: Checking 'loops'.
Information: Checking 'generated_clocks'.
Information: Checking 'signal_level'.
Warning: There are 243 voltage mismatches MAX-MAX - driver rail !=
load rail:

Driver                   Voltage   Load                  Voltage   Margin
-------------------------------------------------------------------------
C_18_ASTlsInst72/OUT     1.08      Multiplier/CSA40/U492/A  1.32  -0.24
C_18_ASTlsInst72/OUT     1.08      Multiplier/CSA40/U656/A1 1.32  -0.24
C_18_ASTlsInst72/OUT     1.08      Multiplier/CSA40/U770/A  1.32  -0.24
Multiplier/S_26_/Q       1.32      GPRs/U7436/A             1.08  -0.24
```

To analyze the operating condition applied to the design paths during timing analysis, it is only possible to check the operating conditions applied on an instance-specific basis by using the `report_cell` command. For example,

```
pt_shell> report_cell ...
...
Cell              Reference     Library             Area  Attributes
-------------------------------------------------------------------
S_reg_reg_27      DFFX2_QQT0    CORE_merge          25.65  n
-------------------------------------------------------------------
Total 1 cell                                        25.65
```

PrimeTime SI supports static multivoltage crosstalk analysis, which deals with the voltage levels of the aggressor and victim nets. For more information, see the *PrimeTime User Guide* and *PrimeTime SI User Guide*.

PrimeTime PX Power Analysis

PrimeTime PX, which is an extension of PrimeTime, performs comprehensive power analysis on gate-level designs. It can perform both average and peak power analysis and can generate detailed power reports.

PrimeTime PX 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 PrimeTime PX, 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 following example is a PrimeTime PX script for multivoltage designs:

```
## READ NETLIST

read_verilog top.v

## LINK THE DESIGN
set link_path "CORE_max_0v70_125c.db"
set link_path_per_instance [list [list {HV_INST} {* CORE_max_1v08.db}]]
lappend link_path_per_instance [list {HV_INST/U100} {* LS.db}]
link_design

## SOURCE UPF COMMANDS
load_upf ./core_upf.tcl

## SOURCE CONSTRAINTS
source ./cstr.tcl

## ANNNOTATE PARASITICS
read_parasitics -keep_capacitive_coupling ./top.spef

## READ SWITCHING ACTIVITY
set_switching_activity [get_nets -hierarchical *] \
  -static 0.1 -toggle 0.1 -period 1
set_switching_activity [get_nets clock] \
  -static 0.5 -toggle 1.0 -period 1

## POWER ANALYSIS
set power_enable_analysis true

create_power_waveforms
report_power
```

The `report_power` command generates many useful power reports, such as peak power and average power reports, for every power domain and power net selected, for either the whole chip or a specific logic hierarchy. The following is a sample power report which lists power consumption by different logic groups.

```
                   Internal  Switching  Leakage    Total
Power Group        Power     Power      Power      Power   (     %)
------------------------------------------------------------------------
io_pad                0.0000    0.0000    0.0000    0.0000 ( 0.00%)
memory                0.0000    0.0000    0.0000    0.0000 ( 0.00%)
black_box             0.0000    0.0000    0.0000    0.0000 ( 0.00%)
clock_network      4.688e-04 1.184e-04 5.424e-07 5.878e-04 ( 5.19%)  i
register           1.522e-03 3.473e-04 5.492e-07 1.870e-03 (16.52%)
```

```
combinational       3.843e-03 4.989e-03 2.637e-05 8.859e-03 (78.28%)
sequential             0.0000    0.0000    0.0000    0.0000 ( 0.00%)

  Net Switching Power  = 5.455e-03   (48.20%)
  Cell Internal Power  = 5.834e-03   (51.55%)
  Cell Leakage Power   = 2.746e-05   ( 0.24%)
                         ---------
Total Power            =    0.0113  (100.00%)
```

For more information about the power analysis flow, see the *PrimeTime PX User Guide*.

PrimeRail Power Network Analysis

PrimeRail extends the Synopsys sign-off solution for power supply network integrity checking. PrimeRail performs voltage drop and electromigration analysis for gate-level and transistor-level designs. It can be used for both static and dynamic power network analysis for a full-chip design.

For a multivoltage design, PrimeRail adds extra value by enabling dynamic power network analysis for a power-up sequence. The tool can also calculate the rush current during a power-up sequence.

Power Network Analysis Flow

The steps in the rail analysis flow depend on the library format used. A CCS power library stores leakage currents and dynamic current waveforms. Therefore, with CCS libraries, there is no need to characterize the library before running cell-level dynamic power and rail analysis. The tool reads the cell power characteristics directly from the CCS power library for transient power and rail analysis.

On the other hand, if NLDM libraries are used, a library should be characterized to capture the models of dynamic current waveforms and intrinsic parasitics using the HSPICE technology built into PrimeRail. While running rail analysis, the tool reads from these precharacterized models attached to the reference library.

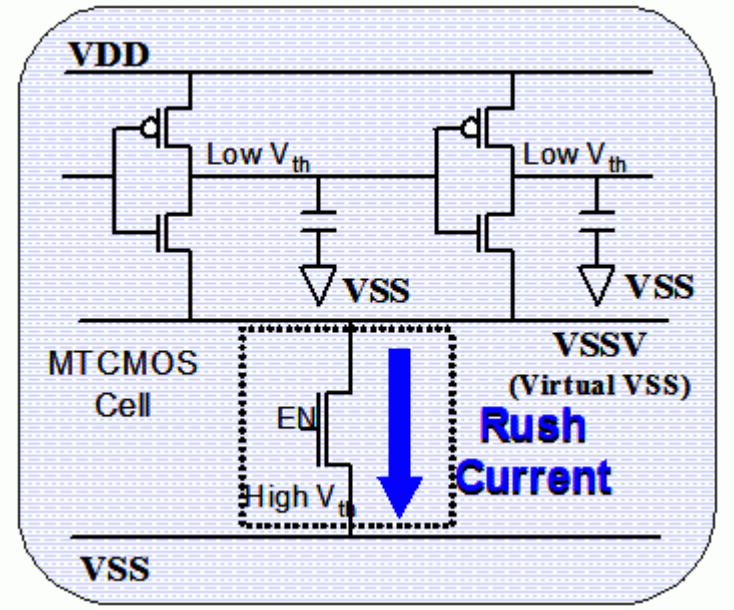
To generate rail analysis maps and current waveforms, you should run a gate-level power analysis to generate the necessary timing and power information. For this purpose, PrimeTime PX is made accessible through PrimeRail. After this step, PrimeRail reads the binary report of power consumption analysis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

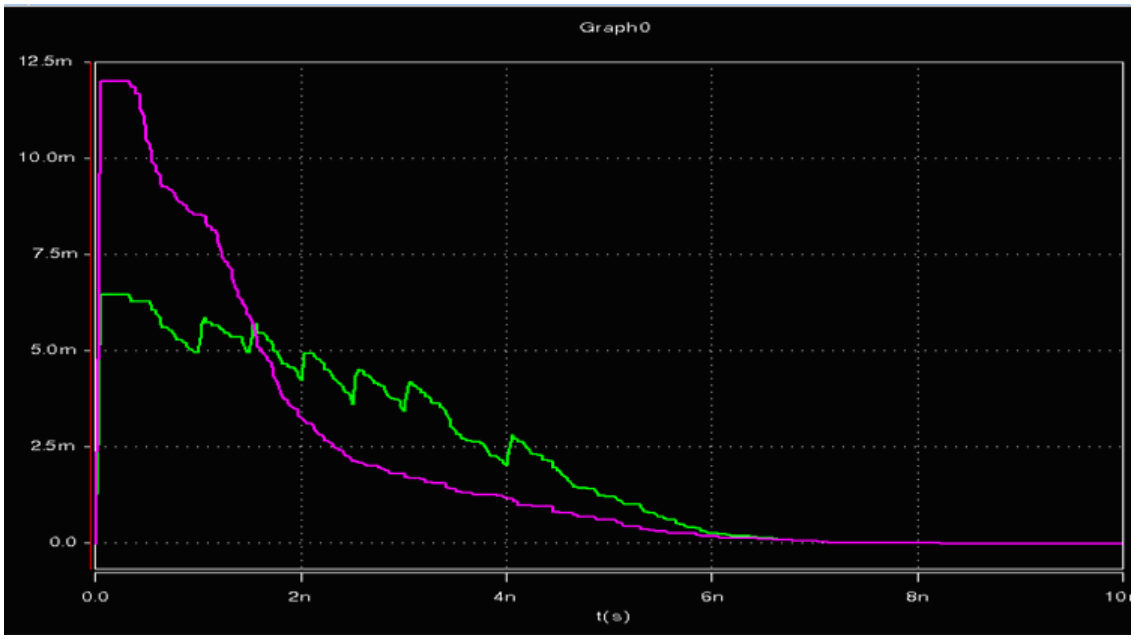
In the coarse-grain, multiple-threshold CMOS (MTCMOS) approach, power switch cells are used to cut off the power supply of an inactive power domain. The number of switch cells in the design can be large and turning them all on or off at the same time can draw a significant rush current from the power grid, as shown in Figure 5-13. 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 5-13 Rush Current Upon Power-Up



PrimeRail provides the capability to analyze the rush current effects of MTCMOS circuits. Figure 5-14 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 5-14 Voltage Drop Waveform With Rush Currents



You can use PrimeRail results to optimize design parameters such as the number of power management cells, their drive strength, and the timing sequence of the control signals.

The flow for power-up sequence analysis is similar to power network analysis flow described previously. After extracting power and ground network, you can perform rush current analysis and wake-up time estimation. PrimeRail supports the display and reporting of rush current, virtual voltage waveforms, leakage current, and wake-up time.

For more information on power network analysis flow, see the *PrimeRail User Guide*.

Glossary

always-on
: The characteristic of a cell, circuit, or power domain that is never powered down. For example, a logic cell used for isolation or retention is an always-on cell. The cell must have separate power supply pins for operation during power-down.

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

clock gating
: A method of reducing power by shutting off clocks to circuits that are not being used.

coarse-grain switching
: A power-switching strategy that switches power on and off for a block as a whole, with all transistors in the block sharing a single power switch.

crowbar current
: The current that flows from the power supply to ground current through a PMOS-NMOS stack during a logic transition, when both the PMOS and NMOS transistors are conducting for a brief period of time.

dynamic power
: Power consumed by CMOS circuits during switching (changing of logic states), consisting of switching power and internal power.

dynamic voltage and frequency scaling
: A speed-versus-power adjustment technique based on changing the supply voltage and operating frequency during operation to meet current workload requirements.

EDA
: Electronic Design Automation, the process of using advanced software tools to design integrated circuits.

enable level shifter
A logic cell that performs both level-shifting and isolation functions, required where power domains can operate at different voltages and can be powered down.

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

fine-grain switching
A power-switching strategy that switches power on and off for individual library cells, thereby allowing the power supply to each cell to be controlled individually.

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

gate leakage
The current that flows between the gate and the source or drain of a transistor, caused by quantum-effect tunneling of electrons through the thin gate insulator.

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

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

isolation cell
A logic cell that can isolate a power-down domain from a domain that is not powered down. The cell operates as a buffer when the input and output sides of the cell are both powered up, but provides a known, constant output signal when the input side is powered down.

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

MTCMOS
Multiple-Threshold CMOS, a feature of a process technology that supports transistors having different threshold voltages. High-threshold transistors can be used where leakage reduction is important, while low-threshold transistors are used where switching speed is important.

multivoltage
The use of two or more different supply voltages on a chip.

NMOS
N-type Metal Oxide Semiconductor, a type of transistor consisting of an N-type source and N-type drain separated by a P-type channel. Applying a positive voltage on the gate induces an N-type conducting channel between the source and drain.

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

PMOS
P-type Metal Oxide Semiconductor, a type of transistor consisting of a P-type source and P-type drain separated by an N-type channel. Applying a negative voltage on the gate induces a P-type conducting channel between the source and drain.

port-punching
The automatic creation of a power supply port by a synthesis or implementation tool, which makes a power connection from one hierarchical level to the next.

port state
A possible state of a power supply port. Each state has a name and an associated condition. The condition can be either a single voltage value, a set of three voltage values (minimum, nominal, and maximum), or the “off” state. The state names are used in power state tables.

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

power domain
A group of elements in the design that share a common power supply distribution network. The set of cells belonging to the power domain is called the extent of the power domain. All of these cells share the same set of power supply rails. The level of hierarchy where the power domain exists is called the scope of the power domain. A power domain must have one primary supply net and one primary ground net, and may optionally have additional supply nets, supply ports, and power switches.

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

power state table
A list of the allowed combinations of voltage values and states of the power switches for all power domains in the design.

power switch
A transistor or transistor array that can switch the power on and off for a portion of the chip, either at the VDD power supply (header switch) or at GND (footer switch), thereby cutting off power to portions of the chip during periods of inactivity. Conceptually, it is a device that turns on and turns off power for a supply net. A switch has an input supply net, an output supply net that can be switched on or off, and at least one input signal to control switching. The switch can optionally have multiple input control signals and one or more output acknowledge signals.

PST
Power state table; a list of the allowed combinations of voltage values and states of the power switches for all power domains in the design.

retention register
A memory register that retains data during power-down periods by keeping the data in a shadow register having an always-on power supply.

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

root cell
A cell defined as belonging to a power domain by the `-elements` option of the `create_power_domain` command, or specified as the scope of the command with the `-scope` option.

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

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

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

static power
The power consumed even when the circuit is not switching, consisting of reverse-biased p-n junction leakage, sub-threshold leakage, and gate leakage.

sub-threshold leakage
The small amount of current that flows between the source and drain of a transistor when the gate voltage is below what is considered the threshold voltage.

supply net
An abstract representation of a power or ground net in the design, representing a contiguous conductor that carries a supply voltage or ground connection.

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

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

switching activity
The relative presence or absence of logic transitions occurring on nets over a period of time, which affects the amount of dynamic power consumed over that period of time.

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

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

UPF
Unified Power Format, a standard set of commands used to specify the low-power design intent for electronic systems. The UPF standard is established by Accellera, an electronics industry standards organization.

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

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

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

Vt
The threshold voltage of a PMOS or NMOS transistor; the gate-to-source voltage at which the transistor begins conducting between the source and drain.

Index

A

C

D

E

F

G

H

I

L

M

N

P

R

S

U

V