Fusion Compiler™ Multivoltage User Guide

Version T-2022.03-SP3, July 2022

SYNOPSYS®

Copyright and Proprietary Information Notice

Destination Control Statement

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

Disclaimer

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

Trademarks

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

Free and Open-Source Licensing Notices

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

Third-Party Links

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

www.synopsys.com

Feedback

Contents

About This User Guide

This guide describes the commands and procedures available in the Fusion Compiler tool to work with multivoltage designs. For more information about the Fusion Compiler tool, see the following companion volumes:

- *Fusion Compiler User Guide*
- *Library Manager User Guide*
- *Fusion Compiler Design Planning User Guide*
- *Fusion Compiler Data Model User Guide*
- *Fusion Compiler Graphical User Interface User Guide*

This user guide is for design engineers who use the Fusion Compiler tool to implement multivoltage designs.

This preface includes the following sections:

New in This Release

Information about new features, enhancements, and changes, known limitations, and resolved Synopsys Technical Action Requests (STARs) is available in the Fusion Compiler Release Notes on the SolvNetPlus site.

Related Products, Publications, and Trademarks

For additional information about the Fusion Compiler tool, see the documentation on the Synopsys SolvNetPlus support site at the following address:

https://solvnetplus.synopsys.com

Feedback

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

- PrimeTime® Suite
- Library Compiler™

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
`Courier`	Indicates syntax, such as `write_file`.
`Courier italic`	Indicates a user-defined value in syntax, such as `write_file` *`design_list`*
`Courier bold`	Indicates user input—text you type verbatim—in examples, such as `prompt>` **`write_file top`**
`Purple`	• Within an example, indicates information of special interest. • Within a command-syntax section, indicates a default, such as `include_enclosing = true | false`
[ ]	Denotes optional arguments in syntax, such as `write_file [-format` *`fmt`*`]`
...	Indicates that arguments can be repeated as many times as needed, such as *`pin1 pin2 ... pinN`*.
|	Indicates a choice among alternatives, such as `low | medium | high`
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Bold	Indicates a graphical user interface (GUI) element that has an action associated with it.
Edit → Copy	Indicates a path to a menu command, such as opening the **Edit** menu and choosing **Copy**.

Convention	Description
Ctrl+C	Indicates a keyboard combination, such as holding down the Ctrl key and pressing C.

Customer Support

Customer support is available through SolvNetPlus.

Accessing SolvNetPlus

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

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

https://solvnetplus.synopsys.com

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

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

Contacting Customer Support

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

1

Multivoltage Design Concepts

A method for reducing power is using multivoltage design. In multivoltage designs, the subdesign instances operate at different voltages. In multisupply designs, the voltages of the various subdesigns are the same, but the blocks can be powered on and off independently. In this user guide, unless otherwise noted, the term multivoltage includes multisupply and mixed multisupply-multivoltage designs.

This section covers the following topics:

Multivoltage and Multisupply Designs

Synopsys design tools support the following types of low-power designs:

- Multivoltage
- Multisupply
- Mixed multivoltage and multisupply

To reduce power consumption, multivoltage designs typically make use of power domains. The blocks of a power domain can be powered up and down, independent of the power state of other power domains (except where a relative always-on relationship exists between two power domains).

Multivoltage designs have nets that cross power domains to connect cells operating at different voltages. Some power domains can be always-on, that is, they are never powered down, while others might be always-on relative to some specific power domain. Some power domains shut down and power up independently, but might require isolation and other special cells. In general, voltage differences are handled by level shifters, which

step the voltage up or down from the input side of the cell to the output side. The isolation cells isolate the power domain. Note that an enable-type level shifter can be used as an isolation cell.

IEEE 1801 UPF Support

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 the power intent throughout the design process. This includes synthesis, physical implementation, and verification.

The UPF file is a script of commands (or constructs) that are defined in the IEEE 1801 standard. These commands specify the power requirements of the design, but they do not specify how the requirements are implemented. The commands specify how to create a power supply network to each design element, the behavior of the supply nets with respect to each other, and how the logic functionality is extended to support dynamic power switching to design elements. The UPF file does not contain any placement or routing information.

UPF files provide a convenient and consistent communication method between different tools in a design flow. For any given UPF command, some tools might act upon the command to perform an operation, while other tools might only read and ignore the command.

The Fusion Compiler tool uses a UPF file to perform synthesis and implementation for multivoltage designs.

Power Intent Concepts

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

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

Each scope in the design has *supply nets* and *supply ports* at the defined hierarchical level of the scope. A *supply net* is a conductor that carries a supply voltage or ground throughout a given power domain. A supply net that spans more than one power domain is

said to be “reused” in multiple domains. A *supply port* is a power supply connection point between two adjacent levels of the design hierarchy, between the parent and child blocks of the hierarchy. A supply net that crosses from one level of the design hierarchy to the next passes through a supply port.

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

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

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

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

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

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

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

Power Domains

Multivoltage designs contain design partitions which have specific power behavior compared to the rest of the design. A power domain is a basic concept in the low-power infrastructure, and it drives many important low-power features across the flow.

By definition, a power domain is a logical grouping of one or more logic hierarchies in a design that share the same power characteristics, including:

- Primary voltage states or voltage ranges (that is, the same operating voltage)
- Process, voltage, and temperature (PVT) operating condition values (all cells of the power domain except level shifters)
- Power net hookup requirements
- Power down control and acknowledge signals, if any
- Power switching style
- Same set or subset of nonlinear delay model (NLDM) target libraries

Thus, a power domain describes a design partition, bounded within logic hierarchies, that has a specific power behavior with respect to the rest of the design. A power domain is strictly a synthesis construct, not a netlist object.

Each power domain has a supply network consisting of supply sets, supply nets, and supply ports and might contain power switches. The supply network is used to specify the power and ground net connections for a power domain. When used together, the power domain and supply network objects allow you to specify the power management intentions of the design.

Every power domain must have one primary power supply and one primary ground. In addition to the primary power and ground nets, a power domain can have any number of additional power supply and ground nets.

A power domain has the following characteristics:

- Name
- Level of hierarchy or scope where the power domain is defined or created
- The set of design elements that comprise the power domain
- Associated set of supply nets that are allowed to be used within the power domain
- Primary power supply and ground nets
- Synthesis strategies for isolation, level-shifters, always-on cells, and retention registers

Feedback

Voltage Areas

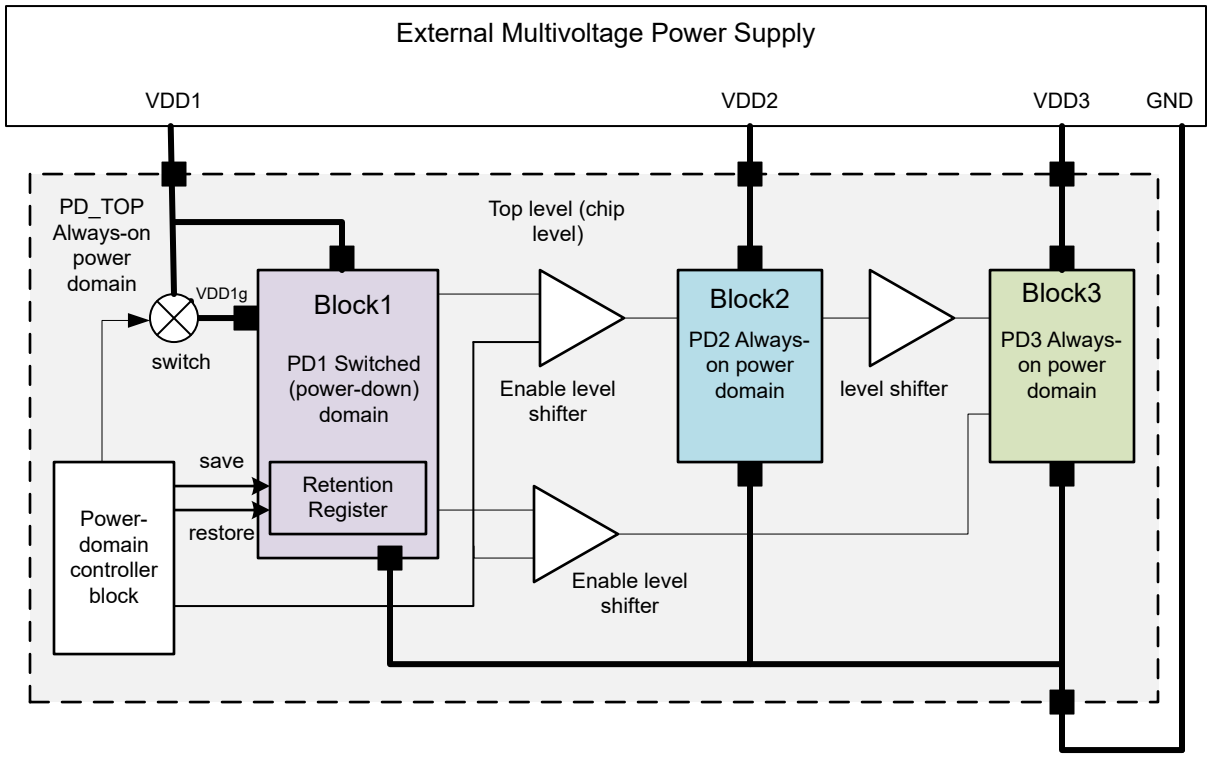
Corresponding to the power domains of logic synthesis, you define voltage areas in physical synthesis as placement areas for the cells of the power domains. Except for level-shifter cells, all cells in a voltage area operate at the same voltage.

There must be a one-to-one relationship between logical power domains and physical voltage areas. A voltage area is the physical implementation of a power domain. A voltage area is associated with a power domain in a unique, tightly bound, one-to-one relationship. A voltage area is the area in which the cells of specific logic hierarchies are the be placed. A single voltage area must correspond to another single power domain, and vice versa. The power domains of a design are defined first in the logical synthesis phase and then the voltage areas are created in the physical implementation phase. All the cells that belong to a given voltage area have the power behavior described by the power domain characteristics.

Power Network Example

The power network example shown in Figure 1 demonstrates some power intent concepts.

Figure 1 Power Intent Specification Example



This chip is designed to operate with three power supplies that are always on, 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.

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 (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.

2

Library Requirements for Multivoltage Designs

To work with multivoltage designs, the logic libraries must conform to the Liberty syntax. The libraries should also contain special cells such as clock-gating cells, level-shifter cells, isolation cells, retention registers, and always-on buffers and inverters. The tool also supports multiple libraries characterized at different voltages.

For more information, see the following topics:

- Liberty PG Pin Syntax
- Power Management Cells
- Suppressing Power Management Cell Insertion
- Power Management Cell Insertion and Bias Supplies

Liberty PG Pin Syntax

In traditional single-voltage designs, all components of the design are connected to a single power supply at all times. Therefore the logic libraries used for synthesizing such designs do not contain details of power supply and ground connections because all the cells are connected to the same type of VDD and VSS.

For multivoltage designs, it is necessary to specify the power supplies that can be connected to specific power pins of a cell. The Liberty syntax supports the specification of power rail connections to the power supply pins of the cells. This power and ground (PG) pin information allows the synthesis tool to optimize the design for power 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 *Advanced Low Power Modeling* chapter in the *Library Compiler User Guide*.

Power Management Cells

Power management cells such as level shifters and isolation cells are not usually part of the original design description. By default, power management cells are automatically inserted during the logic synthesis phase of a design flow. You can also insert them before logic synthesis by running the `create_mv_cells` command. Buffer-type level shifters are

Feedback

inserted during synthesis as part of compilation or by manually instantiating the cells in the RTL.

Multivoltage designs typically use the following types of power management cells:

- Isolation Cells
- Level-Shifter Cells
- Retention Register Cells
- Power-Switch Cells
- Always-On Logic Cells
- Repeater Cells

Name-Based Association of Power Management Cells

Synopsys tools use a specific naming style for power management cells. The name-based associations follow these conventions for power management cells:

- Isolation cells

```
<name_prefix>_snps_<power_domain_name>__<iso_strategy_name>_
   snps_<pin_name>_<inst_index>_<name_suffix>
```

- Level-shifter cells

```
<name_prefix>_snps_<power_domain_name>__<ls_strategy_name>_
   snps_<net_name>_<inst_index>_<name_suffix>
```

- Power switch cells

```
<name_prefix>_snps_<power_domain_name>__<psw_strategy_name>_snps_
   <lib ref name>_R<row_index>_C<col_index>_<inst_index>
```

Requirements of Level-Shifter and Isolation Cells

The following are the requirements of level-shifter and isolation cells:

- Two power supplies.
- Buffer-type and enable-type level-shifter library cells must have the `is_level_shifter` library attribute set to `true`.
- Enable-type level shifters must also have the `level_shifter_enable_pin` library attribute set on the enable pin.
- Isolation library cells must have the `is_isolation_cell` library attribute set to `true`.
- Isolation cells must have the `isolation_cell_enable_pin` library attribute set on the enable pin.

- Level shifters and isolation cells are selected from the target libraries. Therefore, at least one of the libraries must contain these required cells.
- Level-shifter and isolation cells can only be inserted on unidirectional ports.

Inserting Power Management Cells

You can optionally insert power management cells by running the `create_mv_cells` command after loading the power intent with the `load_upf` command.

By default, the command inserts mapped power management cells only if all instances are mapped. If the design includes any unmapped instances, the tool inserts power management cells from the GTECH generic library.

If you use the `-mapped` option with the `create_mv_cells` command, the command inserts cells only from the user-provided logic library and issues an error message if a suitable cell is not available.

To insert multibit power management cells from the logic library, set the `mv.cells.enable_multibit_pm_cells` application option to `true`.

Isolation Cells

Isolation cells are required when a logic signal crosses from a power domain that can 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 when the input side is powered down.

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

An isolation cell is needed when a logic signal crosses from one power domain that can be powered down to one 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 when the input side is powered down.

For more information about creating and using isolation cells and enable level-shifter cells, see the *Library Compiler User Guide*.

This section covers the following topics:

Feedback

- Using NOR-Style Isolation Cells
- Using Single-Rail and Dual-Rail Isolation Cells

Implementing Isolation Cells

The following UPF constructs are associated with isolation cells:

- The `set_isolation` command creates strategies for isolation intent of a power domain.
- The `set_isolation_control` command defines the isolation control signal and also the isolation location.
- The `use_interface_cell` command identifies which library cells to use for each strategy.

Isolation Cell Requirements

The following are the requirements of isolation cells:

- Isolation library cells must have the `is_isolation_cell` library attribute set to `true`.
- Isolation cells must have the `isolation_cell_enable_pin` library attribute set on the enable pin.
- Isolation cells are selected by the tool from the target libraries. Therefore, at least one of the libraries must contain these required cells.
- Isolation cells can only be inserted on unidirectional ports.

Using Latch-Type Isolation Cells

You can insert latch-type isolation cells by specifying the isolation strategy with the `set_isolation` command and `-clamp_value` option, as shown in the following example:

```
fc_shell> set_isolation my_iso ... -clamp_value latch
```

An isolation cell with a clamp value set to `latch` might have an asynchronous set or reset pin. To insert this type of isolation cell with the `set_isolation` command, you must use the `-async_set_reset` option to specify the net name of the asynchronous control signal and its sensitivity (high or low).

For example, the following command creates isolation strategy ISO1 for a latch-type isolation cell. The command specifies that net RS1 is an asynchronous set or reset control signal that is active in the high state:

```
fc_shell> set_isolation ISO1 -domain PD1 \
   -clamp_value latch -async_set_reset {RS1 high}
```

Feedback

You can optionally use the -async_clamp_value option to specify whether the pin is a set input (with an argument of 1) or a reset input (with an argument of 0). The following command specifies that the asynchronous control pin on the isolation cell is a reset pin:

```
fc_shell> set_isolation ISO1 -domain PD1 \
   -clamp_value latch -async_set_reset {RS1 high} \
   -async_clamp_value 0
```

You might not want to specify whether the asynchronous pin should act as a set or reset pin at the time of isolation strategy definition. In this case, use the set_isolation command without the -async_clamp_value option. Then you can later use the set_port_attributes command to set the UPF_async_clamp_value attribute on specific ports outside the isolation strategy to 1 for set or 0 for reset, as shown in the following example:

```
fc_shell> set_port_attributes {port1 port2} \
   -attribute {UPF_async_clamp_value 0}
```

The Fusion Compiler tool executes the set_isolation command options as follows:

- If either or both of the -async_clamp_value or -async_set_reset options are specified for an isolation strategy but the -clamp_value option is not set to latch, the tool issues an error message.
- If a net name is specified with the -async_set_reset option but the sensitivity is not set for that signal, the tool issues an error message.
- If a conflict exists between the -async_clamp_value option of the set_isolation command and the UPF_async_clamp_value attribute for a specific pin, the tool honors the attribute setting and issues a warning.
- If the set_isolation command uses the -async_set_reset option but there is no clamp value set on the specified pin using either the -async_clamp_value option or the UPF_async_clamp_value attribute, the tool ignores the -async_set_reset option with a warning.
- If the -async_set_reset option is not used, the tool ignores and drops the -async_clamp_value option with a warning when the tool executes an action command such as the compile and create_mv_cells commands. In this case, the strategy becomes a normal isolation latch strategy.
- If the -async_set_reset option is not defined for the isolation strategy associated with the port, the tool ignores the UPF_async_clamp_value attribute.

For more information about creating this type of isolation cell, see the *Library Compiler User Guide*.

Feedback

Using NOR-Style Isolation Cells

The tool automatically selects and inserts NOR-style (or clamp-to-zero) isolation cells in the shutdown domain. This isolation cell is a single-rail cell that clamps its output to ground (zero) when the domain's primary supply is turned off. The tool can also optimize dual-rail isolation cells to NOR-style isolation cells to reduce the power and area used.

To specify a NOR-style isolation cell, specify the isolation strategy with the `set_isolation` command and the `-isolation_supply` option. The `-clamp_value` option must be set to zero. For example,

```
fc_shell> set_isolation ISO1 -domain PD_BLK -isolation_supply {} \
   -clamp_value 0 -applies_to outputs
```

Note:

Because the empty isolation supply set `{}` indicates that the isolation cell has no power supply when operating in isolation mode, the tool uses a clamp-to-zero isolation cell (or a NOR-type isolation cell).

When possible, the tool optimizes the design using NOR-type isolation cells. For example, the following command optimizes the implemented isolation cell to use the domain's primary supply (PD_BLK.primary) by using a NOR-type isolation cell:

```
fc_shell> set_isolation ISO1 -domain PD_BLK \
   -isolation_supply {TOP_AO_SS} -clamp_value 0 \
   -applies_to outputs
```

For more details on modeling and using NOR-style isolation cells, see SolvNetPlus article 2370685, "Single-Rail Clamp-to-0 NOR-Type Isolation Cells."

Using Single-Rail and Dual-Rail Isolation Cells

When selecting an isolation cell for mapping, the tool automatically selects single-rail or dual-rail cells based on the isolation cell's rail information and location.

Typically, you use a single-rail isolation cell if the cell is inserted in a domain where the primary rail remains on during shutdown mode. Use a dual-rail isolation cell when the isolation cell needs to be inserted in the shutdown domain and requires a secondary rail connection, so the cell continues to be powered during shutdown mode by a backup supply.

You can create exceptions to this rule by using the `use_interface_cell` command to restrict the availability of cells. The tool can insert a conflicting cell, if the proper cell is not available. For example, if only single-rail cells are available, the tool inserts them even in a shutdown region and issues a warning message.

The tool checks and reports warnings for rail violations, for example, a single-rail isolation cell used in a shutdown domain or a dual-rail isolation cell used in a domain that is powered on more than the shutdown domain.

Feedback

Level-Shifter Cells

In multivoltage designs, a level-shifter cell is usually required when a 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. The level shifter converts a logic signal from one voltage level to another, with a goal of having the smallest possible delay from input to output.

Level shifters can convert from one voltage to a lower voltage, to a higher voltage, or both.

The Fusion Compiler tool supports level-shifter cells with different PG pin configurations as specified by the Library Compiler models:

- Single-rail level shifter
- Dual-rail level shifter with two power pins
- Level shifter with a feedthrough standard cell main rail PG pin that enables the shifting of always-on signals between shutdown power domains
- Level shifter inside a macro cell connected directly to the macro cell's input pins, which eliminates the need to insert external level shifters
- Enable level shifter, which performs both level shifting and isolation functions

For more information about modeling level shifters, see the *Library Compiler User Guide*.

Implementing Level-Shifter Cells

The following UPF commands are used to implement level shifters:

- The `set_level_shifter` command creates a level shifter strategy for a power domain.
- The `use_interface_cell` command identifies available level-shifter library cells for each level-shifter strategy.

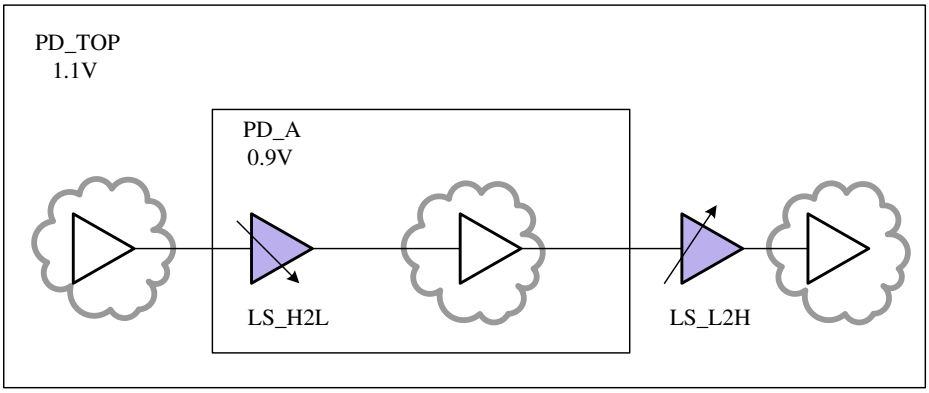
Level-shifter strategy definitions are optional in the UPF. Without a defined strategy, level-shifter cells are automatically inserted based on the voltage definitions in the power state table (PST) defined in the UPF specification. The tool supports both strategy-based and automatic level-shifter insertion.

Figure 2 illustrates the following level-shifter strategies:

```
set_level_shifter LS_H2L -domain PD_A -applies_to inputs \
  -rule high_to_low -location self
set_level_shifter LS_L2H -domain PD_A -applies_to outputs \
  -rule low_to_high -location parent
```

Feedback

Figure 2 Level Shifter Example



User-specified level-shifter strategies have the highest priority; the tool tries to implement as many user-specified strategies as possible.

The tool also attempts to minimize the total number of inserted level shifters.

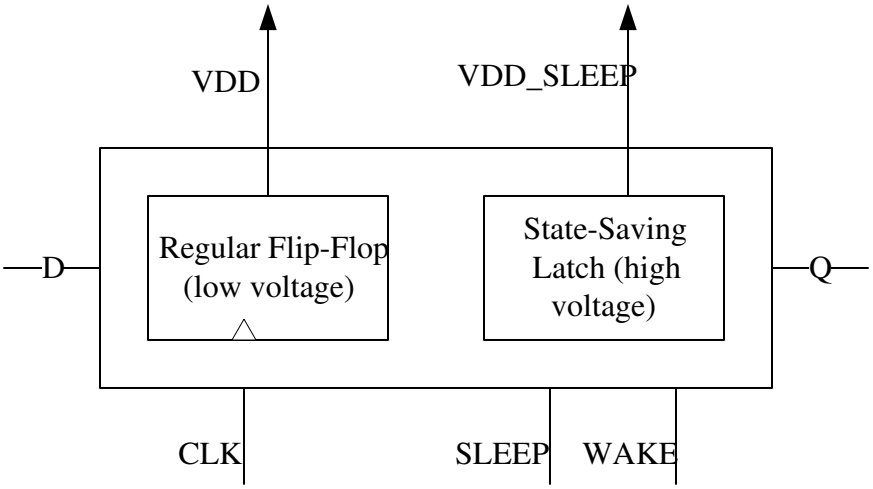
Retention Register Cells

In a design with power switching, one of the ways to save register states before powering down and restoration, is to use retention cells. These cells can maintain their state during power-down by means of a low-leakage register network and an always-on power supply. Retention cells occupy more area than regular flip-flops. These cells continue to consume power when the domain is powered down.

During normal operation of a cell, there is no loss in performance and during power-down mode, the retention cell state is saved. This is possible by using a state-saving latch, which holds the data from the active register (see Figure 3).

Feedback

Figure 3 Retention Register Components



The retention register consists of two separate elements:

- Regular flip-flop or latch

  The regular flip-flop or latch consists of a low-threshold voltage transistors for high performance.

- State-saving latch

  The state-saving latch consists of a balloon circuit modeled with high-threshold voltage transistors. It has a different power supply: VDD_SLEEP.

During the active mode, the regular flip-flop operates at speed and the retention latch does not add to the load at the output. During sleep mode, the Q data is transferred to the state-saving latch, and the power supply to the flip-flop is shut off, thus eliminating the high-leakage standby power. When the circuit is activated with the wake-up signal, the data in the retention latch is transferred to the regular register for continuous operation.

Note:

This example illustrates a two-pin retention register since there are two control signals, sleep and wake.

The Fusion Compiler tool supports the following types of retention registers:

- Two-Pin Retention Registers
- Single-Pin Retention Registers
- Zero-Pin Retention Registers

Feedback

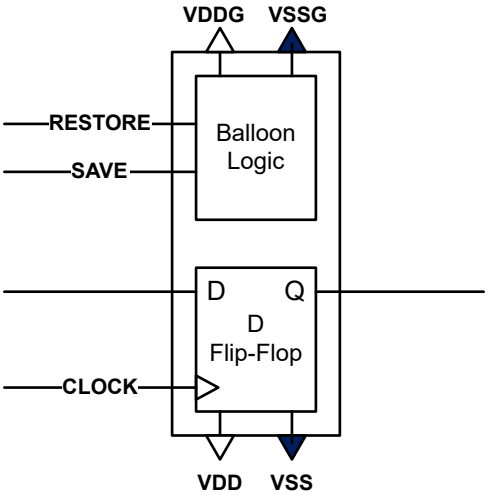
See Also

- Checking Zero-Pin Retention Registers
- Implementing Retention Registers

Two-Pin Retention Registers

You can model a retention register as a two-pin retention register. The two-pin register has two control signals (save and restore), as shown in Figure 4.

Figure 4 Two-Pin Retention Register



Triggering the save pin puts the register in the active mode, in which case the register works as a regular D flip-flop. Triggering the restore pin puts the register in sleep mode, in which case the register works as a state-saving latch.

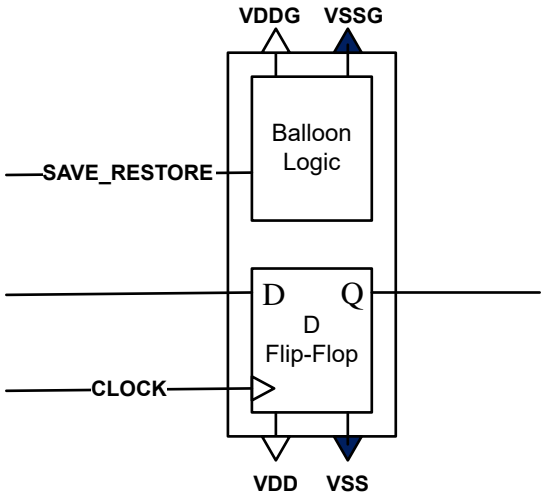
Single-Pin Retention Registers

The tool also supports single-pin retention registers. For single-pin retention registers, the control pin (the save_restore pin), saves and restores the state of the cell depending on the voltage state of the pin. See Figure 5.

In save mode, the single-pin retention register works like a regular latch. In restore mode, it works like a retention cell. That is, the D-input of the register is not passed to the balloon logic and therefore the balloon logic saves the last known value. This value is sent to the Q-output when the restore mode is enabled.

Feedback

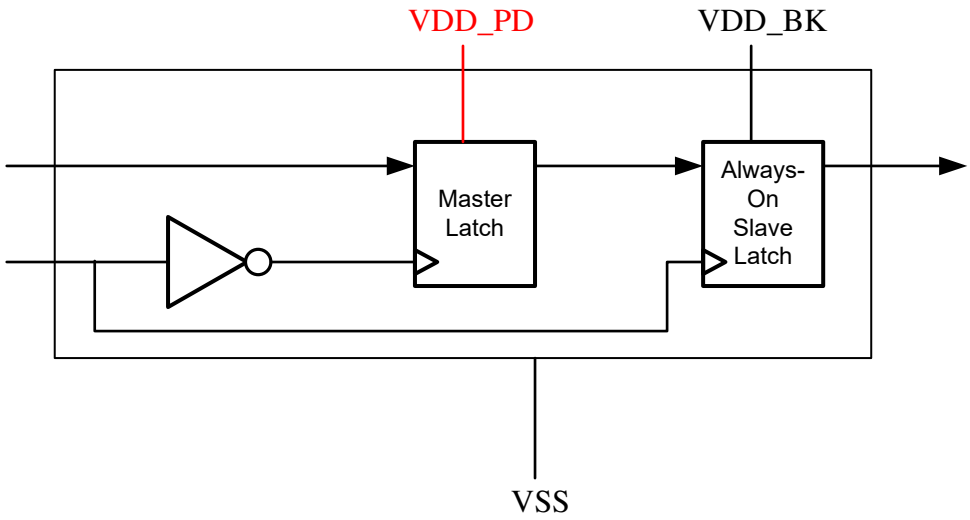
Figure 5 Single-Pin Retention Register



Zero-Pin Retention Registers

The Fusion Compiler tool supports zero-pin retention registers. A zero-pin retention register has no control pins. This type of retention register uses a structure in which the subordinate latch is a low-leakage block with an always-on power supply. Figure 6 shows the internal structure of a zero-pin retention register.

Figure 6 Zero-Pin Retention Register



The subordinate latch maintains the data during both normal operation and shutdown, so no save or restore signal is needed. In this type of retention register, the subordinate latch of the register remains alive during shutdown to maintain the data. Retention occurs when

Feedback

an isolation circuit holds the clock signal in an inactive state. For a rising edge clock, the inactive state is low or zero.

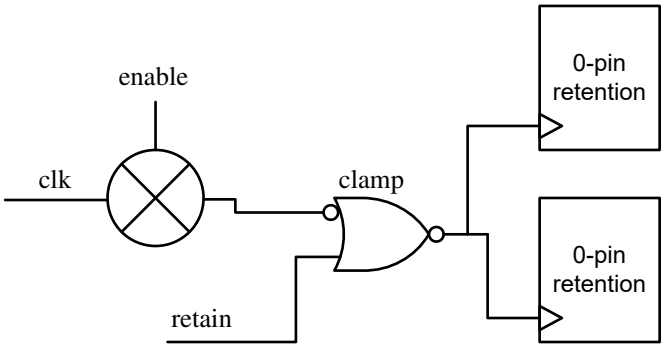
Clamping the Clock Net and Asynchronous Input Pins

To maintain the clock input pins in the inactive state during shutdown, the tool inserts an isolation clamp circuit in the clock network to hold the clock signal in the inactive state for zero-pin retention registers. The clamp cells are placed close to the leaf retention registers.

If you are using retention registers with asynchronous reset or clear input pins to initialize or clear the register contents, these pins must also be prevented from floating during shutdown. The tool inserts an isolation clamp circuit in the asynchronous path to hold the signal while in the inactive state.

Figure 7 shows a typical clock net clamping circuit inserted by the tool. Similar clamping circuits are also inserted on asynchronous pins.

Figure 7 Clock Net Clamping Circuit



By default, the tool buffers the net between the clamp cell and the register's clock or asynchronous pin using an always-on supply. You can disable the buffering using the application option as follows:

```
fc_shell> set_app_options -name  \
   mv.upf.no_buffering_after_retention_clamp_cell -value true
```

Feedback

Checking Zero-Pin Retention Registers

The `check_mv_design` command checks the following:

- Whether the clamp value of the isolation cell on the zero-pin retention register's clock path matches the required condition of the implemented register
- Whether the clamp cells on the zero-pin retention register clock path are powered by an always-on supply
- For dual-rail isolation cells used on the clock path, whether the clamp cell uses an always-on supply (as recommended)

Reporting

To report details on the zero-pin retention registers and their corresponding clamp cells, use the following commands:

- `report_mv_cells -retention`
- `report_mv_cells -retention_clamp`
- `report_mv_cells -retention_clamp -verbose`

Implementing Retention Registers

During the `compile_fusion` command, the Fusion Compiler tool inserts generic retention cells as defined by the UPF specification. These generic cells have the save, restore, and clock pins needed for the final cell mapping.

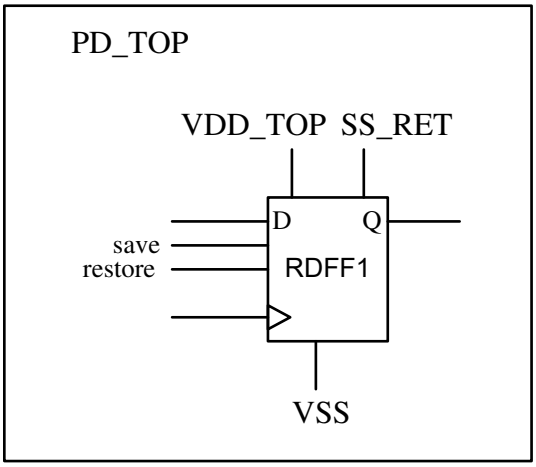
To use retention registers, do the following:

- Ensure that the registers are properly modeled in the Liberty format and compiled to the .db format by the Library Compiler tool. For details, see the *Library Compiler User Guide*.
- Provide a UPF file with a retention strategy that implements the registers using the `set_retention` and `map_retention_cell` commands.
- For zero-pin retention registers, optionally use the `map_retention_clamp_cell` command to specify library cells to use for the clamp cell. This command must not be included in the UPF file.

An example of a two-pin retention register is shown in Figure 8.

Feedback

Figure 8 Two-Pin Retention Register Example

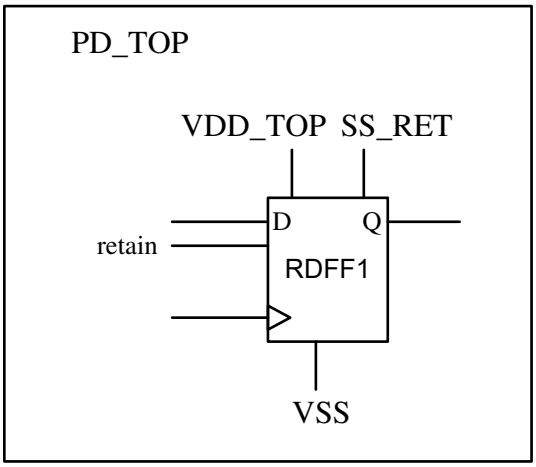


The following UPF command implements the two-pin retention register:

```
fc_shell> set_retention my_ret -domain PD_TOP \
   -retention_supply SS_RET \
   -elements {*RDFF*} \
   -save_signal {save high} \
   -restore_signal {restore high} \
...
```

Figure 9 shows an example of a single-pin retention register implementation.

Figure 9 Single-Pin Retention Register With Balloon Logic



Feedback

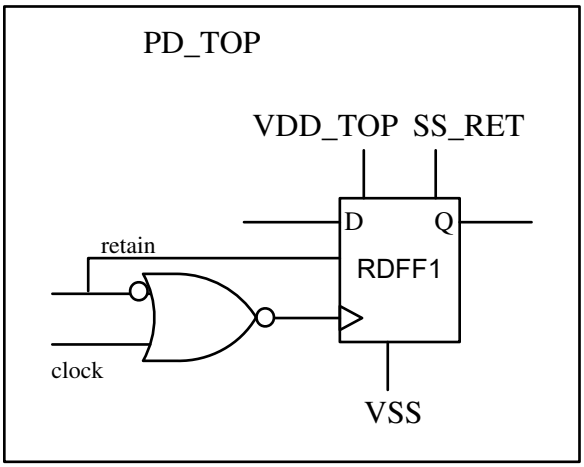
In this example of a single-pin retention register with balloon logic, the pin named retain functions as the save and restore pin. The following UPF command illustrates the implementation for the single-pin retention register:

```
fc_shell>set_retention my_ret -domain PD_TOP \
   -retention_supply SS_RET \
   -elements {*RDFF*} \
   -save_signal {retain high} \
   -restore_signal {retain low} \
   -retention_condition {retain}...
```

The tool reads and saves the `-retention_condition` option, but does not process it. Verification tools use this option.

Figure 10 is an example of implementation of a zero-pin retention register.

Figure 10 Zero-Pin Retention Register



For a zero-pin retention register, there are no explicit `restore_signal` or `save_signal` pins defined, however you still must specify the `-save_signal` and `-restore_signal` options, as follows:

```
fc_shell> set_retention my_ret -domain PD_TOP \
   -retention_supply SS_RET \
   -elements {*RDFF*} \
   -retention_condition {!clock && !reset} \
   -restore_signal {retain low} \
   -save_signal {retain high}
...
```

Power-Switch Cells

In a design with power switching, the power-switch cells provide the supply power for cells that can be powered down. The library description of a power-switch cell specifies the

Feedback

input signal that controls power switching, the pin or pins connected to the power rail, and the pin or pins that provide the virtual or switchable power.

There are two types of power-switch cells, the header type and the footer type. 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.

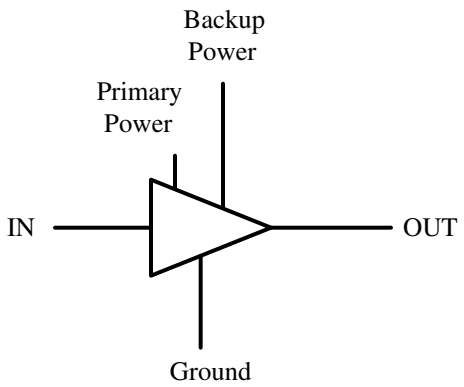
For more information about power-switch cells, see the *Library Compiler User Guide*.

Always-On Logic Cells

Specific cells in shutdown domains might need to continuously stay active, such as 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 in shutdown mode (see Figure 11).

Figure 11 Always-On Logic Cell



The tool performs always-on optimization only when the target library contains always-on inverters and buffers. To use a specific library cell in the optimization of always-on paths, mark the cell with the `always_on` attribute. The tool uses the always-on cells to optimize the always-on paths within shutdown power domains.

For more information about modeling these cells, see the *Library Compiler User Guide*.

Repeater Cells

If the distance between a driver and a receiver is long, a repeater cell might be needed to strengthen the signal. Repeaters are usually simple buffer cells. The UPF provides a way to specify repeaters strategies. Repeater strategies apply to ports of a power domain.

Suppressing Power Management Cell Insertion

Sometimes you might want to suppress power management cell insertion. This might occur because of several reasons:

- The power management cell library model is not available.
- Some power management cells are preinstantiated and the UPF strategies are not final.
- You might want to use the UPF for synthesis, but defer power management cell insertion until implementation.

To suppress power management cell insertion, set the following application options:

```
fc_shell> set_app_options -name mv.upf.skip_all_iso_insertion \
      -value true
fc_shell> set_app_options -name mv.upf.skip_all_ls_insertion \
      -value true
fc_shell> set_app_options -name mv.upf.skip_all_retention_insertion \
       -value true
```

Preinstantiated level-shifter, isolation, and retention cells still exist in the netlist. Preinstantiated GTECH cells of power management cells are not mapped.

The `check_mv_design` command reports missing level shifter and isolation cells for any electrical violation. The tool considers most electrical violations to be error conditions. However, some isolation strategies contain rules that prevent the tool from fixing these violations. The tool issues a warning message instead of an error message in the following cases:

- The `set_level_shifter` command is used with the `-rule low_to_high` or `-rule high_to_low` option, which prevents level shifter insertion.
- The `set_level_shifter` command is used with the `-no_shift` option, which prevents level shifter insertion.

Power Management Cell Insertion and Bias Supplies

Enabling the bias feature affects isolation and retention register cell insertion. Enabling the bias feature does not affect level-shifter cell insertion.

The bias feature affects isolation cell insertion in the following ways:

- To specify an isolation supply using a supply set, use the `set_isolation -isolation_supply` command.
- You can use predefined supply set handles as defined by the IEEE 1801 standard such as `default_isolation`. For instance, for domain PD1, you can use the `PD1.default_isolation` handle.
- You cannot use the `-isolation_power_net` and `-isolation_ground_net` options on bias-enabled domains.
- For an isolation strategy using the `-diff_supply_only`, `-source`, or `-sink` options, bias supplies of driver or load cells are not used to implement the strategy.

The bias feature affects retention register cell insertion in the following ways:

- To specify a retention supply using a supply set, use the `set_retention -retention_supply` command.
- You can use the predefined supply set handle `default_retention`, as defined by the IEEE 1801 standard. For example, `PD1.default_retention` refers to the retention register's default supply.
- You cannot use the `-retention_power_net` and `-retention_ground_net` options on bias-enabled domains.

3

Defining the Power Intent in the UPF

The Fusion Compiler tool supports UPF commands to define, review, and examine the power intent specification. You can use the Fusion Compiler GUI to examine the power intent specification.

This section discusses how to use UPF commands to specify the power intent and how the tool implements the UPF directives.

The section covers the following topics:

Feedback

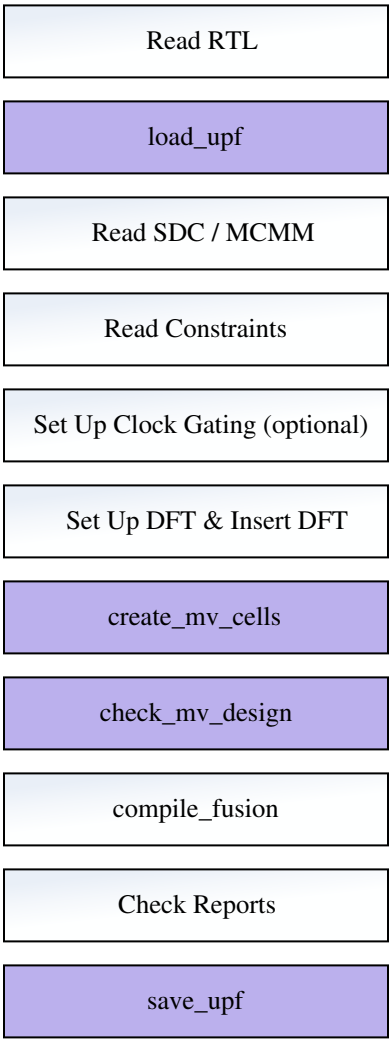
- Specifying Repeater Strategies
- Creating Power Switches
- Power State Tables
- Power Models
- Setting UPF Attributes on Ports and Hierarchical Cells
- Querying for UPF Design and Port Attributes
- Hierarchical Synthesis With ETMs and Macros
- Setting Up Operating Conditions

Multivoltage Design Overview

The Fusion Compiler synthesis flow is shown in Figure 12 with the multivoltage aspects highlighted.

Feedback

Figure 12 Synthesis Flow For Multivoltage Designs



To synthesize a multivoltage design, follow these steps:

1. Read the RTL file.
2. Read the UPF power intent using the `load_upf` command. The command supports the following flows:
   - Traditional (UPF prime) flow
   - Golden UPF flow
3. (Optional) Execute the `create_mv_cells` command. You can skip this and go directly to the `compile_fusion` command. However, using this command might save you

some time and provide some early feedback regarding the feasibility of your power intent. The `create_mv_cells` command does the following:

- Inserts mapped level shifters, isolation cells, and repeaters based on the defined UPF strategies only if all instances are mapped. If the design includes any unmapped instances, the tool inserts power management cells from the GTECH generic library.
- Issues the `commit_upf` command. The `commit_upf` command is optional, but running this command performs some conflict resolution between the power intent and PG net connections.
- Traces power management cells to establish drivers and loads

4. (Optional) Execute the `check_mv_design` command to check the electrical correctness of the design. Although this command is optional, it takes less time to run than the `compile_fusion` command and performs some early checking.
5. Run the `compile_fusion` command to insert mapped power management cells and perform always-on synthesis.
6. Use the `save_upf` command to write out the UPF commands to a specified file. The output UPF file captures tool-derived and user-specified UPF commands. Use the following command for the golden UPF flow:

   ```
   fc_shell> save_upf -format supplemental my_supplemental.upf
   ```

   Use the following command for the UPF prime flow:

   ```
   fc_shell> save_upf my_upf_prime.upf'
   ```

To remove the UPF constraints after reading the UPF file, use the `reset_upf` command. This command removes all UPF constraints and UPF data-dependent constraints from the current design. The design is restored to a default single-voltage UPF condition.

The `reset_upf` command does not affect the netlist. Any commands that might have changed the netlist after the `load_upf` command are not reversed by the `reset_upf` command.

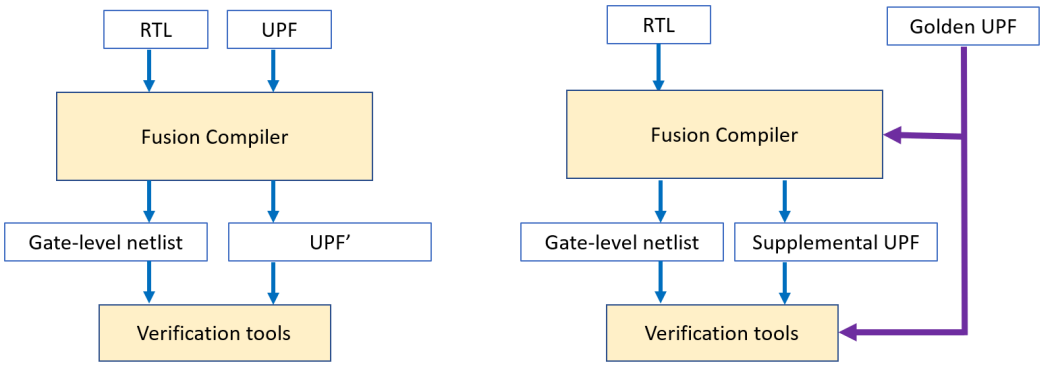
UPF Flows

The Fusion Compiler tool supports both the traditional UPF flow and the golden UPF flow. The golden UPF flow is an optional method of maintaining the UPF multivoltage power intent of the design. It uses the original “golden” UPF file throughout the synthesis, physical implementation, and verification steps, along with supplemental UPF files generated by the Fusion Compiler tool.

Feedback

Figure 13 compares the traditional UPF flow with the golden UPF flow.

Figure 13 UPF-Prime (Traditional) and Golden UPF Flows



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

The golden UPF flow maintains and uses the original UPF file throughout the flow. The Fusion Compiler tool writes power intent changes into a separate supplemental UPF file. Downstream tools and verification tools use a combination of the golden UPF file and the supplemental UPF file, instead of a single UPF' or UPF'' file.

The golden UPF flow offers the following advantages:

- The golden UPF file remains unchanged throughout the flow, which keeps the form, structure, comment lines, and wildcard naming used in the UPF file as originally written.
- You can use tool-specific conditional statements to perform different tasks in different tools. Such statements are lost in the traditional UPF-prime flow.
- Changes to the power intent are easily tracked in the supplemental UPF file.

To enable the golden UPF flow, use the following application option setting before you load the UPF:

```
fc_shell> set_app_options -as_user_default  \
   -list {mv.upf.enable_golden_upf true}
```

To load supplemental UPF files, use the `-supplemental` option with the `load_upf` command.

For more information about using the golden UPF flow, see SolvNetPlus article 1412864, “Golden UPF Flow Application Note.”

Name-Mapping File

Sometimes the synthesis and physical implementation tools change the object names as a result of optimization or ungrouping of objects. In the UPF' flow, these name changes are written out explicitly to the UPF' and UPF'' files. In the golden UPF flow, the tool writes out a supplemental UPF file with only the power intent changes. Object names in the golden UPF file might not match the corresponding objects in the generated Verilog netlist.

Writing the Power Information

The power information updated by the Fusion Compiler tool is saved by using the `save_upf` command.

If you are using the UPF prime mode, the file written by the tool is referred to as the UPF' file. If you are using the golden UPF mode, the tool writes a supplemental UPF file that contains only the changes to the UPF. The original UPF (or golden UPF) is not altered.

The UPF' file (or the combination of the golden UPF and supplemental UPF files) is used as input to downstream tools, such as the PrimeTime, PrimePower, and Formality tools.

The UPF’ file first contains the following information:

- A banner at the top of the file, as shown in the following example:

```
############################################################
#
# Created by ... (R-2020.09) save_upf on ....
#
############################################################
```

- User-specified commands in the order they were provided
- Explicit power connections to special cells such as level shifters and dual supply cells
- Additional UPF commands specified at the command prompt in the Fusion Compiler session. This update is required for the Formality tool to verify the design successfully.

The tool-generated commands are enclosed by the `set derived_upf_true` and `set derived_upf false` statements. Use of this variable and this statement is restricted to the tool. You cannot explicitly set this variable.

Example 1 shows an example of a UPF’ file written by the Fusion Compiler tool.

Feedback

Example 1 UPF File Generated by Fusion Compiler

```
###############################################################################
# Created by ... (R-2020.09) save_upf on Fri ...
#
###############################################################################
###
....
create_power_domain PDT
create_supply_net SN1 -domain PDT
create_supply_net SN2 -domain PDT
create_power_domain PDC -elements {ABC}
create_supply_net SN3 -domain PDC
## Constraints below are generated by Fusion Compiler.
## Please do not modify.
set derived_upf true
if ($derived_upf) {
connect_supply_net SN1 -ports {PORT1}
connect_supply_net SN3 -ports {PORT2}
}
set derived_upf false
```

Customizing UPF Files

To create a UPF file that contains commands only for specific types of cells, use the `-include` or `-exclude` option with the `save_upf` command. These options accept arguments such as `diode_cells`, `pad_cells`, and `physical_only_cells`. See the `save_upf` command man page for the complete list of arguments.

Command filtering for specified cell types applies to the `connect_supply_net`, `set_related_supply_net`, `create_power_domain`, and `set_isolation` commands. You can apply command filtering to both block-level and full-chip UPF files.

Reviewing UPF Specifications

Use the `find_objects` command to find logical hierarchy objects within the specified scope and return the hierarchical names that match the specified criteria. The command returns a null string when nothing matches the specified search pattern.

The Fusion Compiler tool provides a set of query commands that return a Tcl string. Depending on the command and its options, the Tcl string can be a list of design objects or a list of settings for a specific object. If more than one result matches the query at the current scope, the tool issues an error message and returns a 0 result for the Tcl command. Many other Fusion Compiler commands accept Tcl strings as input. You can use the output of the UPF query commands as input to other commands.

These commands do not represent the power intent of a design. Instead, they provide a convenient way to find specific design objects. You cannot use the query commands in a UPF file to be read with the `load_upf` command.

The query commands are as follows:

- `query_cell_instances`

  Returns a list of instances at or below the active scope of the design that use the named cell or module. Duplicate module or cell names that exist in different libraries are not distinguished.

- `query_cell_mapped`

  Returns the reference cell name of the specified cell instance.

- `query_net_ports`

  Returns a list of ports logically connected to the specified net. By default, the command returns only the ports present at the level of the current scope, but you can choose to return leaf cell ports or intermediate hierarchical ports. The command does not report hidden ports or the ports of synthetic netlist objects.

- `query_port_net`

  Returns the name of the net logically connected to the specified port. If no net is connected, the command returns a null string.

- `query_port_state`

  Returns information about port states defined using the `add_port_state` command.

- `query_pst`

  Returns information about power state tables defined with the `create_pst` command.

- `query_pst_state`

  Returns information about power states defined for a specific power state table with the `add_pst_state` command.

- `query_power_switch`

  Returns information about power switches defined with the `create_power_switch` command.

- `query_map_power_switch`

  Returns information about power switch library cells mapped to UPF power switches with the `map_power_switch` command.

For more information, see the command man pages.

Feedback

Naming Rules for UPF Objects and Attributes

The tool verifies the object names created by the UPF commands, so that the names do not conflict with the names of existing objects in the same logic hierarchy. The tool checks the names of ports, power domains, power state tables, power switches, supply sets and nets, or signal nets. Table 1 shows the naming rules applied by the tool for UPF commands.

Table 1 Name Space Rules for UPF Commands

UPF Command Names	Naming Rule
`create_power_domain`, `create_power_switch`, `create_pst`, `create_supply_set`	Within a logic hierarchy, a power domain cannot have the same name as an existing cell, instance, logic port, supply port, logic net, supply net, power switch, power domain, supply set, power state group, or power state table.
`create_logic_net`, `create_supply_net`	Within a logic hierarchy, a net cannot have the same name as an existing cell, instance, supply net, power switch, power domain, supply set, power state group, or power state table.
`create_logic_port`, `create_supply_port`	Within a logic hierarchy, a port cannot have the same name as an existing cell, instance, supply port, power switch, power domain, supply set, power state group, or power state table.
`set_isolation`, `set_level_shifter`, `set_retention`	The isolation, level-shifter, and retention strategies in a power domain must have unique names.

Simple and Hierarchical Names in UPF Commands

The UPF standard requires simple names for option arguments in some UPF commands. By default, the tool enforces this requirement. However, you can allow hierarchical names by setting the `mv.upf.input_enforce_simple_names` application option as follows:

```
fc_shell> set_app_options -name mv.upf.input_enforce_simple_names \
        -value false
```

By default, this application option is `true`.

For existing designs, if the `mv.upf.input_enforce_simple_names` application option is not explicitly set and hierarchical names are used in the design, the tool automatically sets the application option to `false`.

In all cases, the tool writes the output UPF using simple names for those commands that require simple names.

Setting the Scope for UPF Commands

The scope of a UPF command is the level of design hierarchy to which the UPF command applies. The following terms describe aspects of scope definition:

- The *root scope* is the top-level scope in the design hierarchy.
- The *design top instance* is an instance that represents a design for which the power intent has been defined. By default, the design top instance is the top-level power model.
- The *design top module* is the module for which the UPF file that expresses the power intent has been written.
- The *current scope* is an instance that is either the design top instance (represented by a relative path name from the design top instance) or a descendant of the top instance.
- A *design-relative hierarchical name* is interpreted relative to the design top instance by removing the leading slash character (/) and interpreting the remainder as a rooted name in the scope of the current design top instance.

Set the scope in the one of the following ways:

- Use the `load_upf` command with the `-scope` option to specify the new design top instance. The tool changes the current scope, the design top instance, and the design top module, then executes the UPF commands. After completion, the tool restores the design top instance and design top model to their previous values.

  If you use the `load_upf` command without the `-scope` option, the tool executes the UPF commands in the current scope and does not modify the current scope, the design top instance, or the design top module.

- Use the `apply_power_model` command to apply a power model to specified instances. This command implicitly changes the scope of the commands in the power model. After command execution, the current scope is restored to the scope in which the command was invoked.

- Use the `set_scope` command to change the current scope locally. If you use the command without an argument, the scope is the top level of the current design. You can optionally specify an instance name as an argument to change the scope. However, you can only change the scope within the current design subtree. In other words, you cannot set the scope to a scope above the design top instance or below a leaf-level instance.

  The `set_scope` command returns the name of the previous scope as a design-relative hierarchical name.

Feedback

The tool interprets arguments of the `set_scope` command as follows:

- The `set_scope /` command sets the current scope to the current design top instance.
- The `set_scope .` command does not change the current scope.
- The `set_scope ..` command changes the current scope to the parent scope. However, if the current scope is the current design top instance, the current scope is unchanged and the tool issues an error message.

Note:

The `-scope` option of the `create_power_domain` command specifies where to create a new power domain but does not change the current scope for subsequent UPF commands.

Table 2 illustrates the effect of the `set_scope` and `load_upf` commands.

Table 2 Effect of UPF Commands on the Scope

Design top instance before command	Current scope before command	Command	Design top instance after command	Current scope after command
/top	/top/mid	set_scope bot	/top	/top/mid/bot
/top	/top/mid/bot	set_scope .	/top	/top/mid/bot
/top	/top/mid/bot	set_scope ..	/top	/top/mid
//top	/top/mid	set_scope /	/top	/top
/top	/top	load_upf -scope mid	/top/mid	/top/mid
/top/mid	/top/mid	set_scope .	/top/mid	/top/mid
/top/mid	/top/mid	set_scope bot	/top/mid	/top/mid/bot
/top/mid	/top/mid/bot	set_scope /	/top/mid	/top/mid
/top/mid	/top/mid	set_scope ..	n/a (error)	n/a (error)
/top/mid	/top/mid	(after load_upf finished)	/top	/top
/top	/top	load_upf abc.upf	/top	/top
/top	/top	set_scope mid/bot	/top	/top/mid/bot

Feedback

Table 2 Effect of UPF Commands on the Scope (Continued)

Design top instance before command	Current scope before command	Command	Design top instance after command	Current scope after command
/top	/top/mid/bot	set_scope /	/top	/top
/top	/top	load_upf macro.upf -scope macro_inst	/top/macro_inst	/top/macro_inst
/top/macro_inst	/top/macro_inst	set_scope /bot	/top/macro_inst	/top/macro_isnt/bot
/top/macro_inst	/top/macro_inst/bot	set_scope /	/top/macro_inst	/top/macro_inst
/top/macro_inst	/top/macro_inst	set_scope ..	n/a (error)	n/a (error)
/top/macro_inst	/top_macro_inst	(after load_upf finished)	/top	/top
/top	/top	set_scope macro_inst/bot	n/a (error)	n/a (error)

Preserving Logic Port and Logic Net References

In some UPF commands such as the `add_power_state`, `create_power_switch`, and `set_retention` commands, you can reference logic ports and logic nets within a Boolean expression.

When reading the input UPF file, the tool resolves the logic port or logic net using the following precedence, from highest to lowest:

- Bused logic port
- Logic port
- Bused logic net
- Logic net

After resolving the logic port or net inside the Boolean expression, the tool might optimize away the reference port or net unless you set the following application option:

```
fc_shell> set_app_options \
          -name mv.upf.preserve_logic_in_boolean_expr \
          -value true
```

When you set the `mv.upf.preserve_logic_in_boolean_expr` application option to `true`, the tool preserves any logic port or net that might otherwise be removed during optimization. In that case, the tool handles referenced logic ports and logic nets as follows:

- If a referenced logic port is hierarchical, it is not ungrouped
- Referenced logic ports are not removed
- If the referenced object is a logic net, the driving port is preserved

When writing the output UPF file, the tool does the following:

- Writes out an updated Boolean expression that references the same logic port or net reflecting any name changes
- Uses the name of the output net of the pin when updating the Boolean expression, if a logic port was preserved due to a logic net reference
- Excludes any Boolean expression that references logic ports or nets that cannot be updated because the referenced objects were removed

Note:

The `mv.upf.enable_logic_expr_new_processing` application option must be set to `true` (the default) if you want to preserve referenced logic ports and nets.

Creating Power Domains

To create a power domain, use the `create_power_domain` command.

The `-elements` option specifies the list of hierarchical, macro, and pad cells that are added to the extent of the power domain. If the command is specified within the power model of a hard macro, the elements list can also contain named blocks and signal names. The named blocks and signal names must be simple names and not hierarchical names.

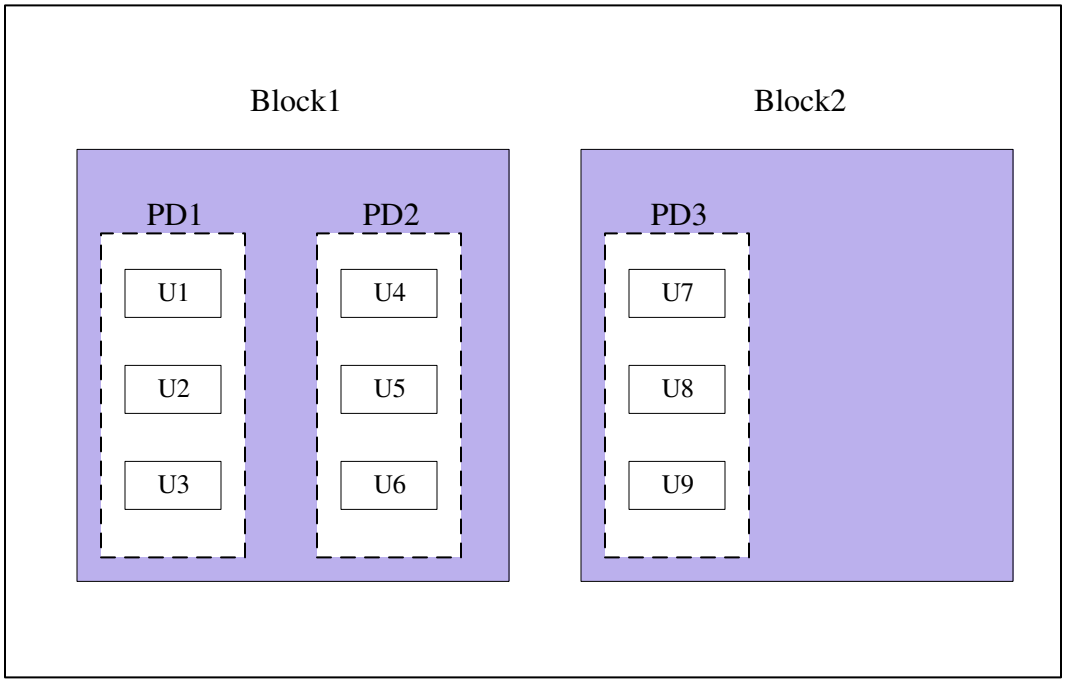
If the required scope is at a lower level than the current scope, use the `-scope` option to specify the name of the instance where the power domain is to be defined.

The UPF standard requires a simple name for the *domain_name* argument.

Figure 14 shows the usage of the `create_power_domain` command.

Feedback

Figure 14 Defining a Power Domain and Scope



To create the PD1 and PD2 power domains, use the following commands:

```
fc_shell> create_power_domain -elements {U1 U2 U3} -scope Block1 PD1
fc_shell> create_power_domain -elements {U4 U5 U6} -scope Block1 PD2
```

Alternatively, you can use the `set_scope` command to set the scope, and then create the power domain, as shown in the following example:

```
fc_shell> set_scope Block1
fc_shell> create_power_domain -elements {U1 U2 U3} PD1
fc_shell> create_power_domain -elements {U4 U5 U6} PD2
```

You can use the `-elements {.}` option if the scope is defined. The `-elements {.}` option includes the current scope as well as all elements in that scope. For example, the following commands create the PD3 power domain in Figure 14:

```
fc_shell> set_scope Block2
fc_shell> create_power_domain PD3 -elements {.}
```

You can define a power domain with an empty elements list and defer the definition of the element list. For example,

```
fc_shell> create_power_domain D1 -elements {}
```

Later, you can add to the elements list using the `-update` option as follows:

```
fc_shell> create_power_domain D1 -elements e1 -update
```

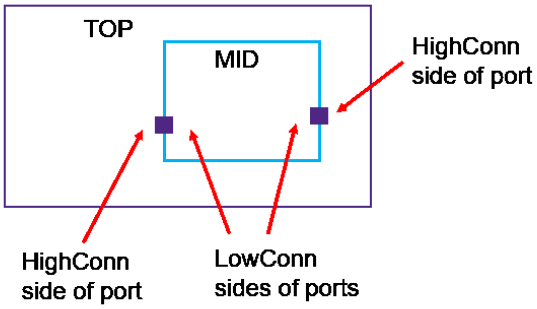
Feedback

Power Domain Boundaries

By default, the tool considers the logical boundary of the root cells of the power domain as the boundary of the power domain. However, to comply with the IEEE 1801 (UPF) standard, the tool can consider a power domain boundary to include the boundary of another domain contained in it. You can specify the elements on the lower-domain boundary for level-shifter and isolation strategy definition, which gives you additional flexibility in selecting the location of the power management cells.

In UPF terminology, a port has two sides: the HighConn side and the LowConn side, as shown in Figure 15. The HighConn side is visible to the parent of the instance whose interface contains the port. The LowConn side is visible inside the instance. In this example, TOP is the parent and MID is the instance.

Figure 15 Sides of a Port



In general, the upper boundary of a power domain is its interface with power domains that are higher in the design hierarchy. You can control how the lower boundary of a power domain is defined.

For more information about power domain boundaries, see Lower-Domain Boundary Support.

Excluding Elements From Power Domains

To exclude elements from a power domain, use the `-exclude_elements` option with the `create_power_domain` command.

For example, the following command creates a power domain that contains cells A and B, but subsequently excludes cell B. The result is a power domain that contains only cell A.

```
fc_shell> create_power_domain -elements {A B} -exclude_elements {B}
```

In the following example, the command creates a power domain that contains cells A and B and all elements in the first level of B's hierarchy (wildcards are supported). The

-exclude_elements option excludes only element B/b1. The result is a power domain whose element list is {A B/b2 B/b3 …} and so on, containing all elements in the first level of hierarchy under B except for b1.

```
fc_shell> create_power_domain -elements {A B/*} -exclude_elements {B/b1}
```

An instance specified in the argument list of the -exclude_elements option must be included in one of the root cells of the power domain (an object in the argument list of an -elements option). If this is not true, the tool issues a warning message and ignores the specified instance. For example, consider the following command:

```
fc_shell> create_power_domain -elements {A B} -exclude_elements {B/b1}
```

In this case, the tool does not exclude element B/b1 because it is not included in the scope of the argument list of the -elements option. The power domain contains only the top levels of cells A and B.

The following conditions apply to the -exclude_elements option:

- The argument list can be empty.
- The tool does ignores duplicate elements in the argument list.
- The argument list cannot contain elements that are not part of the current design.
- A child element does not inherit power domain membership from its parent. For example, consider the following commands:

  ```
  fc_shell> create_power_domain PD_TOP -elements {.}
  fc_shell> create_power_domain PD_A -elements {A A/a*} \
     -exclude_elements {A/a1}
  ```

  The tool reports an error because element A/a1 does not belong to any power domain. The -exclude_elements option excludes A/a1 from the PD_A power domain. In addition, element A/a1 does not belong to the PD_TOP power domain. The {.} argument of the -elements option of the create_power_domain command means that the PD_TOP power domain includes only the logic hierarchy level where the domain is created (the scope).
- The tool ignores duplicate -exclude_elements options.
- The split_constraints command does not write -exclude_elements options because the effective element list is explicitly defined in the generated create_power_domain commands.

When you use the -exclude_elements option with the -update option, the power domain is updated to exclude instances in the -exclude_elements argument list. The tool computes the effective element list after reading all UPF commands. For example, the

following commands create an empty power domain, then make changes to it, resulting in an effective element list of {A}:

```
fc_shell> create_power_domain PD -elements {}
fc_shell> create_power_domain PD -exclude_elements {B} -update
fc_shell> create_power_domain PD -exclude_elements {C} -update
fc_shell> create_power_domain PD -elements {A B C} -update
```

Before you execute a `commit_upf` command, all instances in the design must belong to a power domain.

If a power domain exists in the scope of a model, its excluded elements do not exist and the tool does not issue any warning or error messages.

Viewing Power Domains in the Graphical User Interface

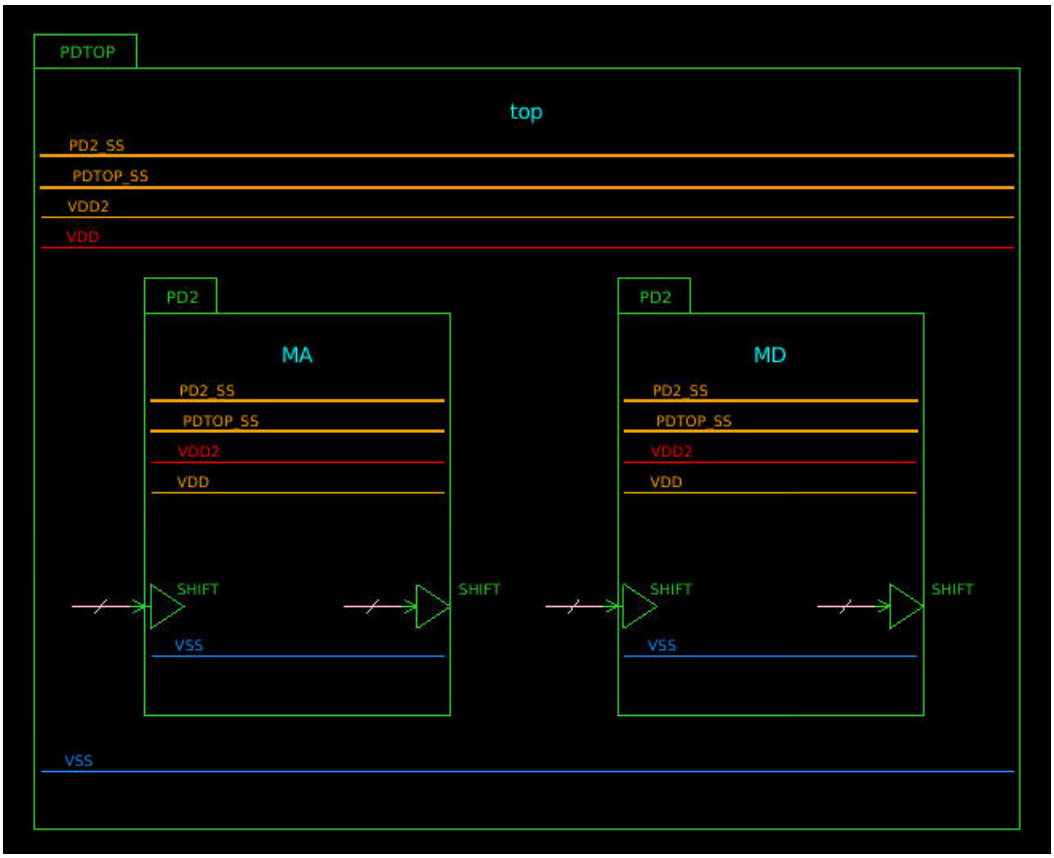
The power domain view (View > Multivoltage Views > Power Domain) displays all power domains that are defined in the current design and its subdesigns. The power domains are organized hierarchically, such that each power domain is located inside its parent power domain.

A power domain is represented by a green colored rectangular bounding box. The name of the power domain is displayed inside the bounding box in the upper left corner of the domain. Figure 16 shows the PDTOP and PD2 power domains and all the UPF objects contained in the power domains. PD2 contains two root cells (or logical modules associated with the domain). These cells are MA and MD.

The size of the power domain symbol varies according to the number and size of the objects that reside within the power domain. The symbol is big enough to display all the UPF objects that are contained in it.

Feedback

Figure 16 Power Domain View in the GUI



Creating Atomic Power Domains

To create an atomic power domain, use the `-atomic` option with the `create_power_domain` command. See the following syntax:

```
create_power_domain name -atomic
  [-elements element_list]
  [-exclude_elements exclude_list]
  […]
```

You must specify the `-atomic` option when you first define the power domain. The tool does not allow updating a non-atomic power domain as atomic, that is, option `-atomic` cannot be specified with `-update`.

However, you can update the extent of an atomic power domain, that is, updating the lists for `-elements` and `-exclude_elements` are allowed.

You can define an empty power domain as atomic. However, you must update the empty power domain with elements before running any action command.

The tool always creates atomic power domains first, independent of the order in which the power domains are defined. Because atomic power domains are processed first, their extents are determined and the tool can identify all the power domains that share extent.

The tool does not allow any instance in the descendant subtree of an atomic power domain to be included in the extent of another power domain, unless:

- the instance name is excluded from the atomic domain or
- the instance name is in the descendant subtree of an instance which is excluded from the atomic domain

The tool has checks in place to verify and report any violations for the preceding cases.

Note:

It is not mandatory to add an instance excluded from a power domain as the root cell of another power domain. There can be cases where the power domain cannot be determined for a few elements in the design. The tool provides an error message for such elements.

Examples

Example 2

```
create_power_domain PD_TOP
 -elements {m2 m1/b1} -include_scope
create_power_domain PD_MID -atomic
 -elements {m1}
 -exclude_elements {m1/b1}
```

In this example, PD_MID is an atomic power domain which is created first. Excluding m1/b1 from the atomic domain allows it to be used in PD_TOP.

Example 3

```
create_power_domain PD_EMPTY -elements {} -atomic
```

You can define an empty power domain as atomic, as in this example. Before you run any action command, you must ensure to update the empty domain so that the domain contains elements.

Example 4

```
create_power_domain PD_TOP -elements {.} -include_scope
          -exclude_elements {i_core} -atomic

create_power_domain PD_CORE -elements {i_core}
commit_upf
```

As shown in this example, excluding the element on which PD_CORE will be created from atomic PD_TOP is allowed.

Reporting Atomic Power Domains

You can use `report_power_domain` in your design to check which power domains are defined as atomic.

The following is an example of the `report_power_domain` output for an atomic power domain:

```
****************************************
Report : Power Domain
Design : top
****************************************
------------------------------------------------------------------------------
Power Domain             :  PD_CORE (atomic)
Current Scope            :  / (top scope)
Elements                 :  core1
Voltage Area             :  DEFAULT_VA
Available Supply Nets    :  VDD_CORE, VDD_ISO, VDD_REG, VDDtop, VSStop
Available Supply Sets    :  SS_CORE, SS_ISO, SS_REG_BANK, SS_TOP

Default Supplies            - Power -     - Ground -
  Primary                :  VDD_CORE [0.95, switchable]
                                          VSStop [0.00]
  Isolation              :  --            --
  Retention              :  --            --
…
```

Hierarchical Flow Support for Atomic Power Domains

This topic discusses the hierarchical flow support for atomic power domains with examples.

Top-Down Hierarchical Flow

There is no special handling of atomic power domains in the top-down hierarchical flow. The tool uses the existing methodology of characterizing power domains for characterizing atomic power domains.

Feedback

The following rules apply:

- If the atomic power domain is defined at or below the scope that gets characterized, then the atomic power domain is characterized to the BLOCK as is.
- If the cell being characterized is the root cell of an atomic power domain:
  - Domain split happens and an atomic power domain with same name is characterized to BLOCK.
  - As is the case currently for non-atomic power domains, no explicit `merge_domain` attribute is needed.
- If the cell being characterized is not a root cell of an atomic power domain:
  - Domain split happens and an atomic power domain with same name is characterized to BLOCK.
  - The `merge_domain` attribute is set for this cell in TOP UPF.

Bottom-Up Hierarchical Flow

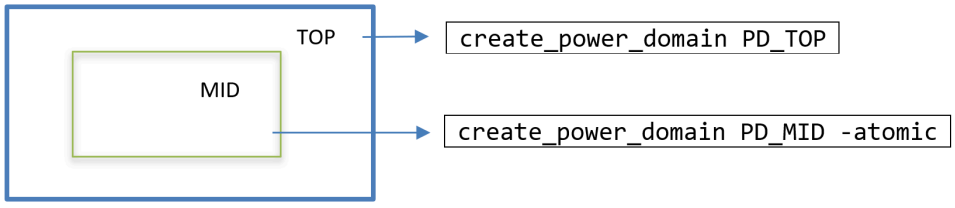
In the bottom-up hierarchical flow, the tool propagates constraints from BLOCK to TOP design. There are two scenarios with respect to the handling of power domains.

1. Non-Domain Merging

In this scenario, atomic power domains defined in BLOCK are propagated to BLOCK scope. If this propagation violates the rule that elements in the extent of an atomic power domain cannot be defined as the root cell of another power domain, the tool reports a warning message to indicate this violation.

Success Case Example

Consider the following example:



In this case, during constraint propagation, the atomic power domain from BLOCK PD_MID gets propagated to the TOP design. The following syntax shows the full UPF post constraint propagation:

```
create_power_domain PD_TOP -include_scope
```

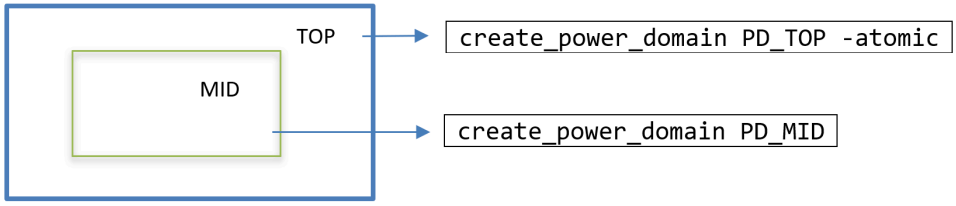
Feedback

```
create_power_domain PD_MID -elements mid_inst -scope mid_inst -atomic
```

This full UPF does not violate any rule of atomic power domains. So, during the execution of the synthesis command, the tool does not generate any warning or error messages related to atomic power domains.

Error Case Example

Consider another example:



In this case, during constraint propagation, the non-atomic power domain from BLOCK PD_MID gets propagated to the TOP design. The following syntax shows the full UPF post constraint propagation:

```
create_power_domain PD_TOP -include_scope -atomic

create_power_domain PD_MID -elements mid_inst -scope mid_inst
```

This full UPF violates the rule (of atomic power domains) that elements in the extent of an atomic power domain cannot be defined as the root cell of another power domain. The element mid_inst, part of atomic power domain PD_TOP (it is not excluded in PD_TOP), is the root cell of a non-atomic power domain mid_inst/PD_MID. During constraint propagation, the tool reports the following warning message and continues the flow:

```
Warning: Element mid_inst in the extent of atomic power domain PD_TOP has
been defined as root cell of another power domain mid_inst/PD_MID
```

You must fix the UPF by excluding mid_inst from the atomic power domain PD_TOP. The tool then allows mid_inst to be the root cell of mid_inst/PD_MID.

2. Domain Merging

The following table describes the four possible combinations of TOP and BLOCK power domains, for domain merging during constraint propagation, and the tool behavior in all these combinations:

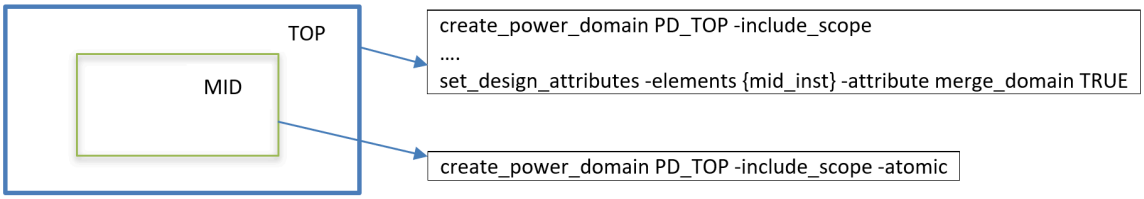
	Non-atomic power domain in BLOCK	Atomic power domain in BLOCK

Feedback

Non-atomic power domain in TOP	Existing behavior	Domain merging fails; both TOP and BLOCK power domains are retained in the final UPF. See Scenario 1: TOP is Non-Atomic and BLOCK is Atomic.
Atomic power domain in TOP	Domain merging fails; both TOP and BLOCK power domains are retained in the final UPF. See Scenario 2: TOP is Atomic and BLOCK is Non-Atomic.	• Matching domains: Only TOP atomic power domain is retained in the final UPF. See Scenario 3: TOP and BLOCK are Atomic and Domain Merging Succeeds. • Non-matching domains: Domain merging fails; both TOP and BLOCK power domains are retained in the final UPF. See Scenario 4: TOP and BLOCK are Atomic and Domain Merging Fails.

Scenario 1: TOP is Non-Atomic and BLOCK is Atomic

Consider the following example:



In this scenario, during constraint propagation, the tool compares the atomic power domain from BLOCK with the non-atomic power domain in the TOP design and the power domains do not match. So, domain merging fails with the following message and the tool retains both the domains:

```
Error: Unable to merge domain PD_TOP and mid_inst/PD_TOP because
mid_inst/PD_TOP is atomic but PD_TOP is not atomic. (UPF-168)
```

The following is the full UPF post constraint propagation:

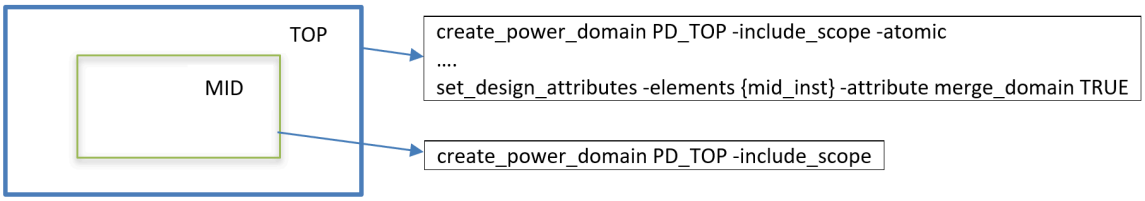
```
create_power_domain PD_TOP -include_scope
create_power_domain PD_TOP -elements {mid_inst} -scope mid_inst -atomic
…
set_design_attributes -elements {mid_inst} -attribute merge_domain TRUE
```

Since the tool generated an error message during constraint propagation, you must update the UPF before proceeding with the flow.

Feedback

Scenario 2: TOP is Atomic and BLOCK is Non-Atomic

Consider the following example:



In this scenario, during constraint propagation, the tool compares the non-atomic power domain from BLOCK with the atomic power domain in the TOP design and the power domains do not match. So, domain merging fails with the following message and the tool retains both the domains:

```
Error: Unable to merge domain PD_TOP and mid_inst/PD_TOP because PD_TOP
is atomic but mid_inst/PD_TOP is not atomic. (UPF-168)
```

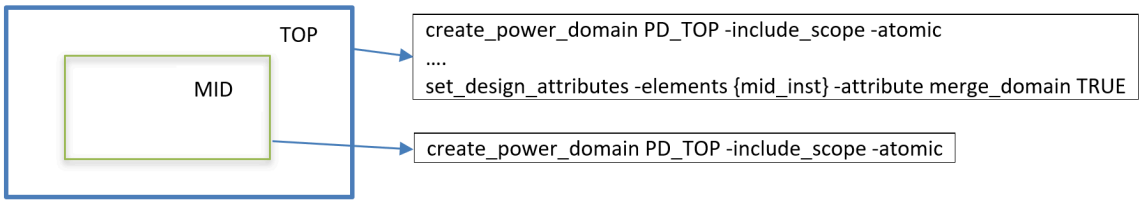
The following is the full UPF post constraint propagation:

```
create_power_domain PD_TOP -include_scope -atomic
create_power_domain PD_TOP -elements {mid_inst} -scope mid_inst
…
set_design_attributes -elements {mid_inst} -attribute merge_domain TRUE
```

In this scenario too, since the tool generated an error message during constraint propagation, you must update the UPF before proceeding with the flow.

Scenario 3: TOP and BLOCK are Atomic and Domain Merging Succeeds

Consider the following example:



In this scenario, during constraint propagation, the tool compares the atomic power domain from BLOCK with the atomic power domain in the TOP design and the power domains and say all their properties match. So, domain merging succeeds and the tool drops the domain from BLOCK.

The following is the full UPF post constraint propagation:

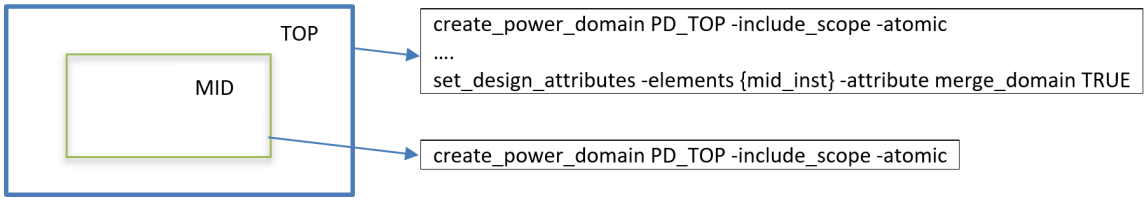
```
create_power_domain PD_TOP -include_scope -atomic
```

```
…
set_design_attributes -elements {mid_inst} -attribute merge_domain TRUE
```

This full UPF does not violate any rule of atomic power domains. So, during the execution of the checker/synthesis commands, the tool does not generate any warning/error messages related to atomic power domains.

Scenario 4: TOP and BLOCK are Atomic and Domain Merging Fails

Consider the following example:



In this scenario, during constraint propagation, the tool compares the atomic power domain from BLOCK with the atomic power domain in the TOP design and say some of their properties do not match. So, domain merging fails and the tool retains both the domains.

The following is the full UPF post constraint propagation:

```
create_power_domain PD_TOP -include_scope -atomic
create_power_domain PD_TOP -elements {mid_inst} -scope mid_inst -atomic
…
set_design_attributes -elements {mid_inst} -attribute merge_domain TRUE
```

In this scenario too, since there is an error during constraint propagation, you must update the UPF before proceeding with the flow.

Multiple Power Domains in a Single Voltage Area

Within a single voltage area, power domains can be shared if they have equivalent primary supplies. Power domains can be combined if the primary supplies meet any of the following conditions:

- The supplies are connected in the UPF intent
- The supplies are electrically equivalent (physically connected)
- The supplies are functionally equivalent (have equivalent power state tables)

To specify that supplies are electrically equivalent, use the `connect_supply_net`, `create_supply_set -update`, `set_equivalent`, or `associate_supply_set` commands. To share power domains, use the `set_design_attributes` command to specify a list

of power domains for the `shared_voltage_area` attribute. In any set of shared power domains, one power domain is defined as the primary power domain.

One power switch can be implemented for each shared voltage area. Use the `create_power_switch`, `create_power_switch_array`, or `create_power_switch_ring` command as appropriate for the design. Then use the `connect_power_switch` command to connect power switch cells based on the `shared_voltage_area` attribute.

The control signals (sleep and acknowledge or ack signals) of power switches can be defined in the RTL (preferred) or in UPF commands such as the `create_logic_port` or `connect_logic_net` commands.

The `connect_power_switch` command implements power switches based on the strategy defined for the primary power domain. The `-connect_sleep_and_ack_in_shared_domains` option specifies to reuse existing connections between power switches if the connections already exist in the netlist. With this option, the tool connects and shorts the sleep inputs and acknowledge (ack) outputs in shared domains. For daisy-chain connections, the acknowledge output of the primary domain must be specified with the `-ack_port` option; it will be connected to and drive the control inputs and acknowledge outputs of other domains in the daisy chain. To identify the sleep and acknowledge ports of the subordinate power domains, use the `-control_port` and `-ack_port` options of the `connect_power_switch` command.

If it is necessary to specify which sleep or acknowledge pins of fine-grained macros are related in a daisy chain, use the `-fine_grained_lib_pins` option to list specific pins. This option must be used with the `-connect_sleep_and_ack_in_shared_domains` option.

Feedback

Creating Supply Ports

To create the power supply and ground ports, use the `create_supply_port` command.

The name of the supply port should be a simple (not hierarchical) name and unique at the level of hierarchy it is defined. Unless the `-domain` option is specified, the port is created in the current scope or level of hierarchy and all power domains in the current scope can use the created port.

You can optionally use the `-direction` option of the `create_supply_port` command to specify the port direction. The valid values are as follows:

- `in` specifies an input port (the default).
- `out` specifies an output port.
- `inout` specifies an input/output port.
- `internal` specifies an internal port. Use this value for ports that connect virtual nets in the design. If a UPF supply net only connects to leaf ports with the internal direction, the tool recognizes that this net should not exist in the physical implementation. An example usage is to connect block supplies that are known to switch together. Using the `internal` designation allows you to provide a name that can then be referenced by other commands such as the `connect_supply_net` or `find_objects` commands.

The following examples show how to create the ports in Figure 1. The following commands create supply ports VDD1, VDD2, VDD3, and GND at the top level of the design hierarchy (PD_TOP):

```
fc_shell> create_supply_port VDD1
fc_shell> create_supply_port VDD2
fc_shell> create_supply_port VDD3
fc_shell> create_supply_port GND
```

The following commands create supply ports VDD1, VDD1g, and GND in power domain PD1:

```
fc_shell> create_supply_port VDD1 -domain PD1
fc_shell> create_supply_port VDD1g -domain PD1
fc_shell> create_supply_port GND -domain PD1
```

The following commands create supply ports VDD2 and GND in power domain PD2 and VDD3 and GND in power domain PD3:

```
fc_shell> create_supply_port VDD2 -domain PD2
fc_shell> create_supply_port GND -domain PD2
fc_shell> create_supply_port VDD3 -domain PD3
fc_shell> create_supply_port GND -domain PD3
```

The `create_supply_port` command automatically creates a domain-independent supply net with the same name as the supply port. The new supply net is connected to the port on the side lower in the hierarchy. If a supply net with the same name already exists within the same power scope, the tool does not create a new supply net.

For example, the `create_supply_port VSS` command achieves the same result as the following commands:

```
fc_shell> create_supply_net VSS
fc_shell> create_supply_port VSS
fc_shell> connect_supply_net VSS -ports VSS
```

After an implicit supply net is created, the net and the port that created it do not retain an association. You can perform operations on the supply port and implicit supply net independently, such as renaming or deleting the objects.

Adding Port State Information to Supply Ports

The `add_port_state` command adds state information to a supply port. This command specifies the name of the supply port and the possible states of the port. The first state specified is the default state of the supply port. The port name can be a hierarchical name. Each state is specified as a pair of arguments: the state name and the voltage level for that state. The voltage level can be specified as a single nominal value, set of three values (minimum, nominal, and maximum), or 0.0, or the keyword off to indicate the off state. The state names are also used to define all possible operating states in the power state table.

Supply states specified at different supply ports are shared in a group of supply nets and supply ports directly connected together. However, this sharing does not happen across a power switch.

The following command is an example of the definition of states for power nets:

```
fc_shell> add_port_state header_sw/VDD -state {HV 0.99} \
   -state {LV 0.792} -state {OFF off}
```

The following command is an example of the definition of states for ground nets:

```
fc_shell> add_port_state footer_sw/VSS -state {LV 0.0} \
   -state {OFF off}
```

The following command defines two states (HV1 and HV2) with the same voltage (1.2 V) on the VDD1 supply port. The duplicate port states are useful in a hierarchical flow, where the top level and block-level ports have different state names but the same voltage.

```
fc_shell> add_port_state VDD1 -state {HV1 1.2} -state {HV2 1.2}
```

Feedback

Creating Supply Nets

A supply net connects supply ports or supply pins. To create a supply net, use the `create_supply_net` command. The UPF standard requires a simple name for the *net_name* argument.

The supply net is created in the same scope or logic hierarchy as the specified power domain. When you use the `-reuse` option, the specified supply net is not created; instead an existing supply net with the specified name is reused.

```
fc_shell> create_supply_net GND_NET -domain PD1
fc_shell> create_supply_net GND_NET -domain PD2 -reuse
```

When a supply net is created, it is not considered a primary power supply or ground net. To make a specific power supply or ground net of a power domain, the primary supply or ground net, use the `set_domain_supply_net` command.

All supply nets, including the ground, must be assigned an operating voltage. If any supply net does not have an assigned operating voltage, the tool issues a `UPF-057` error message during the execution of the `compile_fusion` command. Before compiling the design, use the `check_mv_design -pg_netlist` command to ensure that operating voltages are defined for all the supply nets.

The operating voltage that you have already set cannot be removed. However, you can override the existing settings by using the `set_voltage` command again.

When the `create_supply_port` command is used, the Fusion Compiler tool automatically creates a supply net connected to the port and with the same name as the port. If you subsequently use the `create_supply_net` command and specify the same net name, the tool checks to determine if the automatically-generated net is used. If the automatically-generated net is not used, the tool creates the manually-specified net. Otherwise, the tool issues an error message about a naming conflict.

The tool considers an automatically-generated net to be in use in the following conditions:

- The net is referenced in the power or ground supply of an isolation or retention strategy.
- The net is referenced in the output or input supply of a power switch.
- The net is referenced in the primary power or ground supply of a domain.
- A supply state is set on the net.
- The net is part of a supply set.
- The net is connected to a PG pin or supply port other than the supply port that generated it.

Feedback

- The net is connected to a power state table supply.
- The net is set as equivalent to another supply net.
- The net is associated with ports or pins by the `set_related_supply_net` command.

Using Custom Resolution Functions

You can use a custom resolution function when using the `create_supply_net` command. The tool supports custom resolution functions defined in a package used by the verification tools. A custom resolution function implies that multiple drivers on a net are allowed. The tool reads and saves the function name in the UPF and checks for conflicting functions applied to the same net, but otherwise does not use the function information.

To use this feature, use the `-resolve` option and the name of the resolution function, as follows:

```
fc_shell> create_supply_net VDD -resolve my_package::my_resolution
```

Specifying Primary Supply Nets for a Power Domain

To define the primary power supply net and primary ground net for a power domain, use the `set_domain_supply_net` command.

Every power domain must have one primary power and one ground connection. When a supply net is created it is not a primary supply net. You must use the `set_domain_supply_net` command to designate the specific supply net as the primary supply net for the power domain. All cells in a power domain are assumed to be connected to the primary power and ground net of the power domain. If the power or ground pins of a cell in a power domain, is not explicitly connected to any supply net, the power or ground pin of the cell is assumed to be connected to the primary power or ground net of the power domain to which the cell belongs.

The following example shows the commands to specify VDD and GND nets as the primary power and ground net of the PD_TOP power domain.

```
fc_shell> set_domain_supply_net -primary_power_net VDD \
  -primary_ground_net GND PD_TOP
```

Note:

If you use supply sets to define the primary supply and ground, the supply nets that you specify must belong to the same supply set. For more information, see Specifying Supply Sets.

Feedback

Connecting Supply Nets

The `connect_supply_net` command connects the supply net to the specified supply ports or pins. The connection can be within the same level of hierarchy or to ports or pins down the hierarchy.

You can also use the `connect_supply_net` command to connect to the internal PG pins of macro cells containing fine-grained switches.

The UPF standard requires a simple name for the *supply_net_name* argument.

The following example shows the use of the `connect_supply_net` command to connect supply nets to various supply ports at different levels of hierarchy or power domains.

```
fc_shell> connect_supply_net GND_NET -ports GND
fc_shell> connect_supply_net GND_NET -ports {B1/GND B2/GND B3/GND} GND
```

You can also use the function of a supply set with the `connect_supply_net` command, as shown in the following example:

```
fc_shell> create_supply_set ss
fc_shell> connect_supply_net ss.ground -ports {B1/GND}
```

Use the `create_supply_net -resolve parallel` command when

- A supply net connects to the internal PG pins of more than one macro cell with a fine-grained switch.
- A supply net is associated with a supply set group that has multiple drivers at the scope of the supply net; the net should be created using the `create_supply_net -resolve parallel` option.

Note:

The `connect_supply_net` command ignores connections to the pins of physical-only cells.

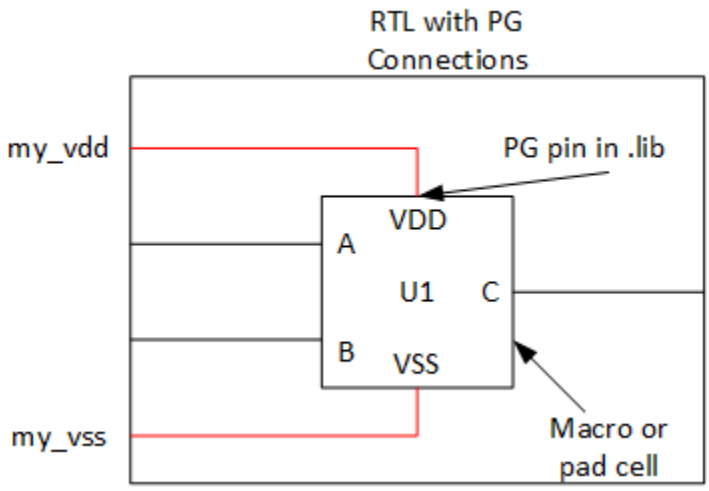
Supporting RTL Designs With PG Connections

If the RTL design contains PG nets and pin connections to macros, I/O, and power management cells, the tool allows this RTL to be used in a UPF flow. To use this feature, you should create matching nets in the UPF. The tool matches the UPF nets to the RTL PG nets. Then the tool creates the required ports and makes the connections in the UPF.

Consider the example in Figure 17.

Feedback

Figure 17 Example of RTL With PG Connections



In order for the tool to allow the PG net in the RTL, include the following command in the UPF specification:

```
create_supply_net my_vdd
```

The tool then writes the following connections in the output UPF:

```
create_supply_port my_vdd
connect_supply_net my_vdd -ports {my_vdd}
connect_supply_net my_vdd -ports {U1/VDD}
```

You must also specify the my_vss supply net in a similar manner.

Preparing RTL Designs With PG Connections

To prepare an RTL design with PG connections and UPF specifications for synthesis, do the following:

1. Read the RTL that has the PG connectivity.
2. Load the UPF specification with the `load_upf` command. The UPF should have the PG nets defined that are present in the RTL.
3. Run the `commit_upf` command to convert the RTL PG information to UPF.
4. Resolve all the MV-165 error messages. These are issued if there is a mismatch between the UPF and RTL with regard to PG connectivity.
5. Reload the UPF and rerun the `commit_upf` command. The tool should successfully infer the PG connections.

Feedback

6. Set voltages on the supply nets using the `set_voltage` command.
7. Optimize the design.

Resolving PG Connection Conflicts

Table 3 describes how the tool resolves conflicts between the input UPF specification and the RTL with PG connectivity.

Table 3 RTL PG Connectivity and UPF Conflict Resolution

User-Specified Input UPF	Derived Connection from RTL PG	Resolution
connect_supply_net VDD \ -ports {macro_inst/VDD}	connect_supply_net VDD \ -ports {macro_inst/VDD}	Duplicate connections on ports. Duplicate supply nets are allowed.
connect_supply_net VDD \ -ports {macro_inst/VDD}	connect_supply_net VDD1 \ -ports {macro_inst/VDD}	Input UPF takes precedence over the RTL. The input UPF is used.
No connection specified	connect_supply_net VDD1 \ -ports {macro_inst/VDD}	New connections to ports are added to the generated UPF created by the tool.

Inferring and Resolving Supply Nets With PG Netlists

To infer or resolve supply net connections in the UPF based on the existing net connections in the PG netlist, use the `infer_supply_from_pg_net` command after the UPF is loaded, but before committing the UPF. If you have already committed the UPF, run the `commit_upf` command again, because supply inference might add new `create_supply_net` commands to the UPF.

The tool issues an error message if there is an explicit call to the `connect_supply_net` command or a tool-derived `connect_supply_net` command in the UPF, but the PG netlist contains a conflicting connection. To override these errors, use the `mv.pg.continue_infer_pg_with_conflict` application option. Setting this application option to `true` gives preference to the explicit or tool-derived UPF connections, and the tool issues a warning message to inform you that the UPF has preference over the netlist.

By default, resolution of PG nets occurs when you use the `commit_upf` command, unless you specify the `commit_upf -skip_resolve_pg_net` option or set the `mv.upf.auto_resolve_pg_net` application option to `false`.

Displaying Supply Nets in the Supply Network View

In the supply network view, a supply net is represented by a line or a line segment. Different colors are used to differentiate the type of the net, as shown in Table 4.

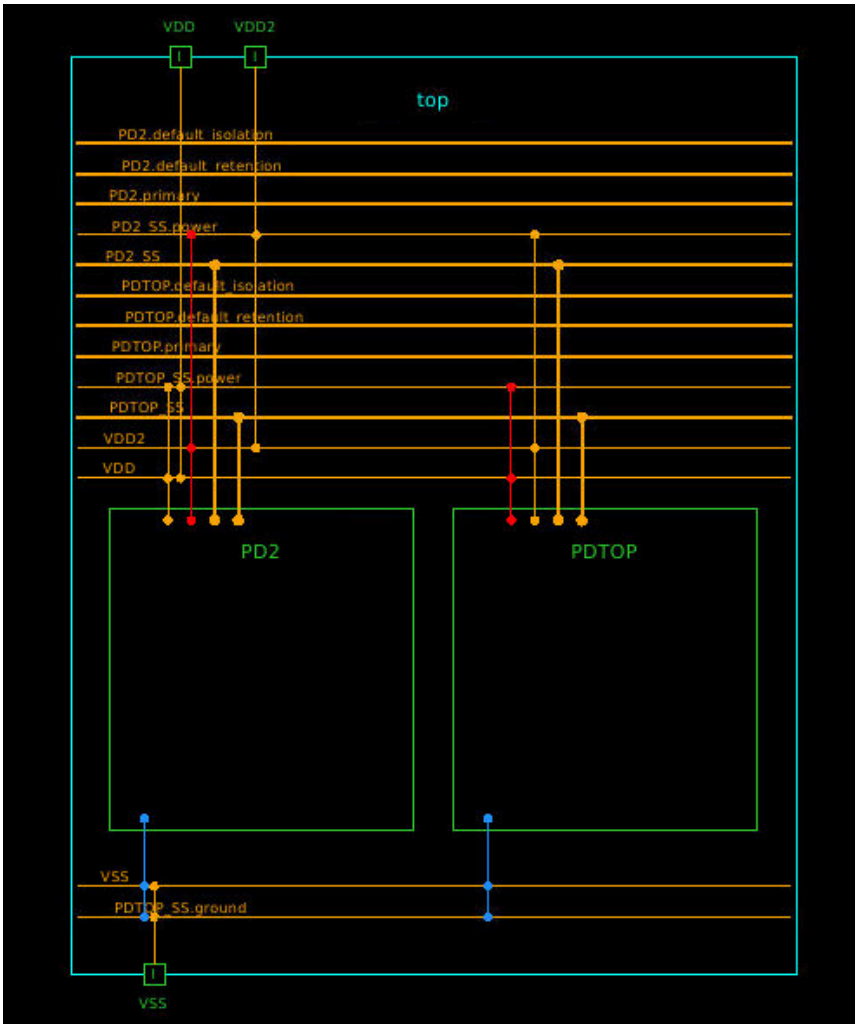
Feedback

Table 4 Colors Used to Represent Types of Net Segments

Color	Net Segment
Red	Primary power net
Blue	Primary ground net
Yellow	All other net segments

As shown in Figure 18, the supply network view displays all the supply nets in the current design and the current design's subdesigns, and their supply net connections.

Figure 18 Power Supply Nets in the Supply Network View



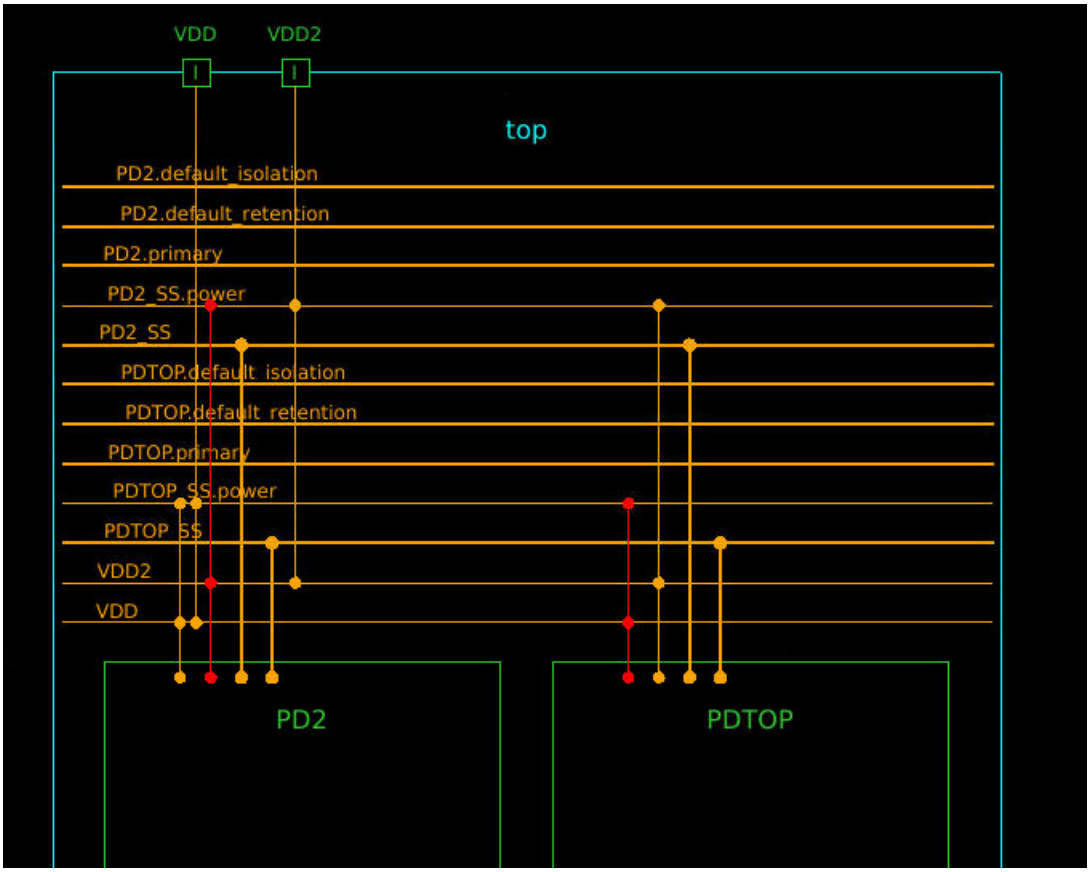
Feedback

The location of the supply nets in the diagram is based on the location of the power domains where they belong and also on the type of the supply net. Each power domain that a supply net belongs to contains a line segment indicating the supply net.

Horizontal segments represent supply nets inside the power domain. Vertical segments represent nets that are reused in multiple power domains and that are connected to another object, such as a supply port or a power switch.

Power supplies extend down from the top of the power domain, and ground nets extend up from the bottom of the power domain. Figure 19 and Figure 20 show an expanded view of the connection to the power supplies and ground.

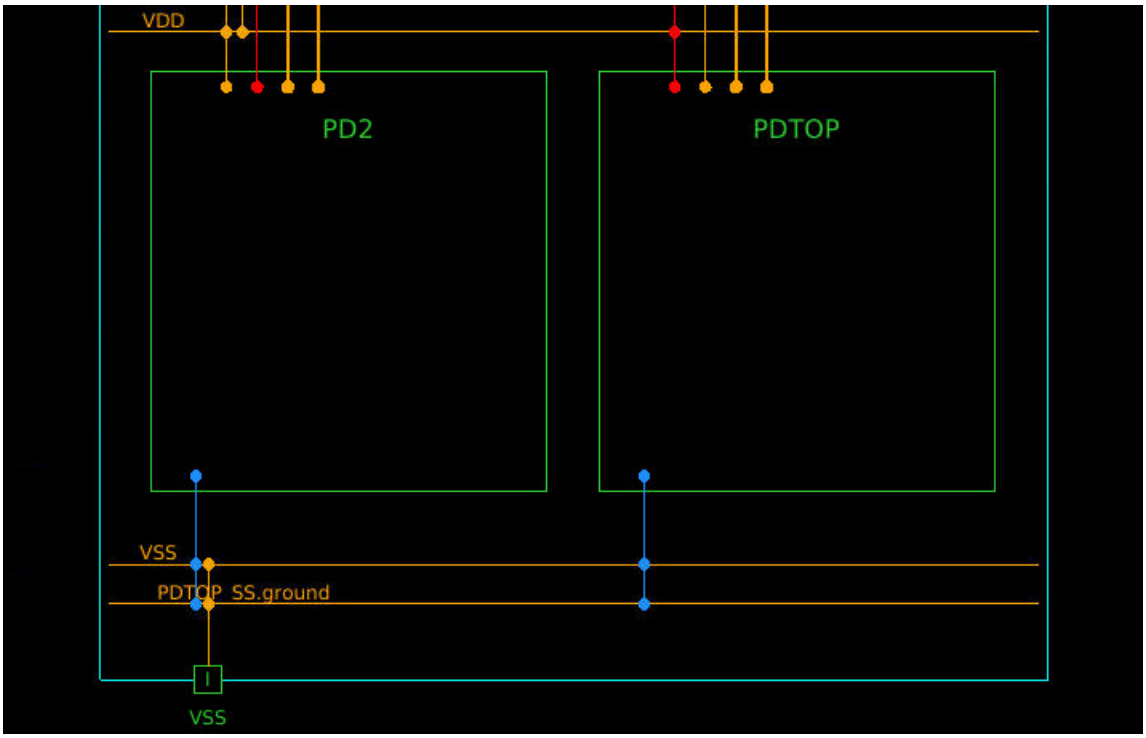
Figure 19 Power Supply Connections in the Supply Network View



In Figure 19, the VDD2 net is the primary supply net of the PD2 power domain. However, it is not the primary supply net of the power domain TOP. VDD is the primary supply net of domain PDTOP.

Feedback

Figure 20 Ground Connections in the Supply Network View



In Figure 20, VSS is the primary ground net of both the power domains PD2 and PDTOP.

Specifying Supply Sets

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

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

A supply set consists of the following functions:

- Power
- Ground

You can access the functions of the supply set by using the name of the supply set and the name of the function. To access the power function of the supply set *SS*, specify *SS.power*. To access the ground function of the supply set *SS*, specify *SS.ground*.

Creating Supply Sets

To create a supply set, use the `create_supply_set` command. The supply set is created in the current logic hierarchy or the scope.

The UPF standard requires a simple name for the *supply_set_name* argument.

The following example shows how to create a supply set and associate it with the primary power supply of a power domain:

```
fc_shell> create_supply_set primary_supply_set
fc_shell> create_power_domain PD_TOP
fc_shell> set_domain_supply_net PD_TOP \
  -primary_power_net primary_supply_set.power \
  -primary_ground_net primary_supply_set.ground
```

Note:

When you specify a supply set as the primary power and ground supply of the power domain, both the primary and the ground supply must belong to the same supply set.

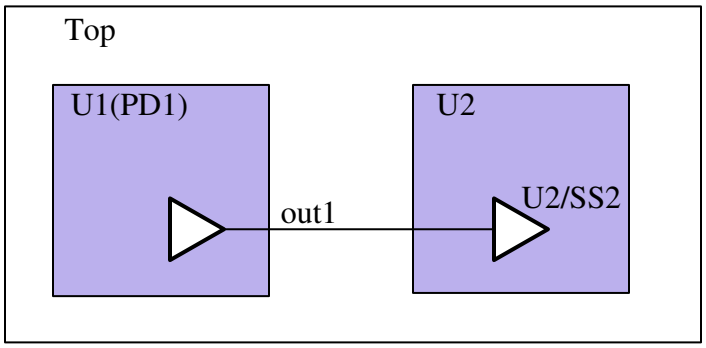
Reference-Only Supply Sets and Supply Nets

When using the hierarchical flow and separating blocks, you might have supply sets or nets that reside outside the current block. In order to refer to these supply sets or nets outside the block after you run the `split_constraints` command, the tool creates a reference-only supply set or supply net. This supply is meant only to resolve supply references in strategies when using the hierarchical flow. You cannot use the supply to power actual cells.

For the example in Figure 21, the U2 power domain has an isolation strategy that refers to a local supply in U2 as a sink supply. If you split U1 into a separate block, the UPF for U1 would need to replace the U2/SS2 in the U1 UPF since U2/SS2 is not valid anymore when the UPF for each block is separated. Meanwhile, the real supply that powers U2/SS2 is only available to power cells in U2 and is not visible outside of U2. After you integrate the design with U1, there needs to be a way to reestablish U2/SS2 to whatever it is replaced with in U1. To do this, the tool creates a reference-only supply for U1.

Feedback

Figure 21 Reference-Only Supply Set Example




```
create_power_domain Top
set_scope U2
create_supply_set SS2
set_scope
create_power_domain PD1 -scope U1 -elements U1
set_isolation iso -domain U1/PD1 -sink U2/SS2
```

To create a reference-only supply, the tool creates the supply set and then marks it as reference-only using the design attribute `reference_only`. For example,

```
create_supply_set SS2'
set_design_attributes -elements . -attribute reference_only {SS2'}
```

The power states of U2/SS2 are also copied to SS2' in the UPF for U2. The receiver supply of the out1 port is also set to SS2' as follows:

```
set_port_attributes -elements out1 -receiver_supply SS2'
```

These additions to the U1 UPF ensure that when U1 is optimized separately, it has all the information for correct multivoltage cell insertion in U1. In the Top UPF, a `set_equivalent` command is added to establish the relationship between SS2’ and U2/SS2:

```
set_equivalent -sets {U1/SS2' U2/SS2}
```

Because the `reference_only` attribute can also apply to supply nets, the tool must add the power state table behavior of the original supply net to the reference-only supply net. To do this, the tool adds the `add_supply_state` command to the block UPF.

Note:

The `add_supply_state` command can only be used with the `add_pst_state` command and not the `add_power_state` command.

Feedback

Creating Supply Set Handles

When you create a power domain, the following supply set handles are created by default:

- `primary`
- `default_isolation`
- `default_retention`

In addition to these predefined supply set handles, you can define supply set handles by using the `-supply` option of the `create_power_domain` command. To associate multiple supply sets with a power domain, use the `-supply` option multiple times.

Supply set handles are created at the scope of the power domain and are available for use in the power domains that are at the same or lower scope than the power domains where they are created. Use the following naming convention to refer to a supply set handle: *power_domain_name.supply_set_handle*. When a power domain is deleted, its supply set handles are also deleted.

You can also specify the `extra_supplies_#` keyword with the `-supply` option of the `create_power_domain` command to restrict the availability of the supplies in the power domain. For more information about using the `extra_supplies_#` keyword, see Restricting Supply Sets.

The following script example shows how to create a power domain and associate a supply set with the power domain:

```
# Create the supply sets
create_supply_set primary_supply_set
# Create power domain and associate it with the supply set
create_power_domain PD1 -supply {primary primary_supply_set}
```

Restricting Supply Sets

Supply sets are domain-independent and can only be updated with domain-independent nets. To restrict the supply sets available to a power domain, use the `extra_supplies_#` keyword with the `-supply` option of the `create_power_domain` command, as shown in the following example:

```
fc_shell> create_power_domain SUB_DOMAIN \
   -supply {extra_supplies_1 supply_set1} \
   -supply {extra_supplies_2 supply_set2} -elements mid1/PD_MID
```

Alternatively, if you do not want the power domain to use extra supply nets other than those that are already defined in other strategies, specify the `extra_supplies ""`

keyword (without the index) with the `-supply` option of the `create_power_domain` command, as shown in the following example:

```
fc_shell> create_power_domain PD_MID -scope mid1 \
   -supply {extra_supplies ""}
```

It is an error to use both the `extra_supplies_#` and `extra_supplies ""` keywords simultaneously.

By default, a power domain can use domain-independent supply nets and supply nets defined in the power domain. However, when you define supply sets with the `extra_supplies_#` keyword, the power domain is restricted to use

- The primary supply of the power domain
- The supplies listed as `extra_supplies_#`
- The supplies specified by the isolation strategy of the power domain
- The supplies specified by the retention strategy of the power domain
- The supplies defined or reused as domain-dependent supplies in the power domain

Refining Supply Sets

To redefine the functions of a supply set, use the `-update` option of the `create_supply_set` command. You must use the `-update` and the `-function` options together, to associate the function names with the supply nets or ports.

The following example uses the `-update` option to associate supply nets to the functions of the supply set:

```
fc_shell> create_power_domain PD_TOP
fc_shell> create_supply_net TOP_VDD
fc_shell> create_supply_net TOP_VSS
fc_shell> create_supply_set supply_set \
  -function {power TOP_VDD} \
  -function {ground TOP_VSS} \
  -update
```

The following rules apply when you update a supply set with a supply net:

- Voltage rule

  The voltage of the supply set handle must match with the voltage of the supply net with which the supply set is updated.

  If voltage is not specified for the supply net, then after the update, the voltage on the supply set handle is inferred as the voltage of the supply net.

- Function rule

  The supply set function must match with the function of the supply net with which the supply set is updated.

  The tool issues an error message when,

  - The ground handle of a supply set is used to update power handle of another supply set and vice versa.
  - The supply net updated with the ground handle of a supply set is connected to a power supply port or pin of a power object, such as a power domain, and vice versa.
  - Scope rule

  The scope of supply set must match with the scope of the explicit supply net with which the supply set is updated.

- Availability rule

  The explicit supply net with which the supply set is updated, must be domain-independent.

- Connection rule

  The explicit supply net with which the supply set is updated, should not be connected to a driver port when the supply set handle is connected to a driver port unless a resolution function is defined for the explicit supply net.

- Conflicting supply state names rule

  A supply set handle cannot be updated with a supply net or supply set if their power states are not identical.

- Valid power-state-table rule

  When a number of supply sets are updated to the same supply net, only one supply set can be present in the power state table.

Defining the Power States for a Supply Set

Power states are attributes of a supply set. Using the `add_power_state` command, you can define one power state for all those supply nets of the supply set that always occur together. For each power state of the supply set, you must use one `add_power_state` command. By default, the undefined power states are considered illegal states. You can define multiple power states for a supply set to form a power state table.

For more information on power states and power state tables, see Power State Tables.

Feedback

Associating Supply Sets

Using the `associate_supply_set` command, you can associate a supply set with another predefined supply set or supply set handle. When two supply sets are associated, the tool considers the two supply sets to be connected, and their functions resolve to the same supply nets.

Using the `-handle` is optional. The following commands are equivalent:

```
associate_supply_set SS1 -handle PD1.primary
associate_supply_set {SS1 PD1.primary}
```

The `associate_supply_set` command accepts either simple or hierarchical names and accepts a list of supplies to associate.

You can associate two or more supply sets as follows:

```
fc_shell> associate_supply_set {SS1 SS2 mid/SS3}
```

Each of the supply set functions (power, ground, nwell, and pwell) of the supply sets are treated as the same supply net or connected supply net.

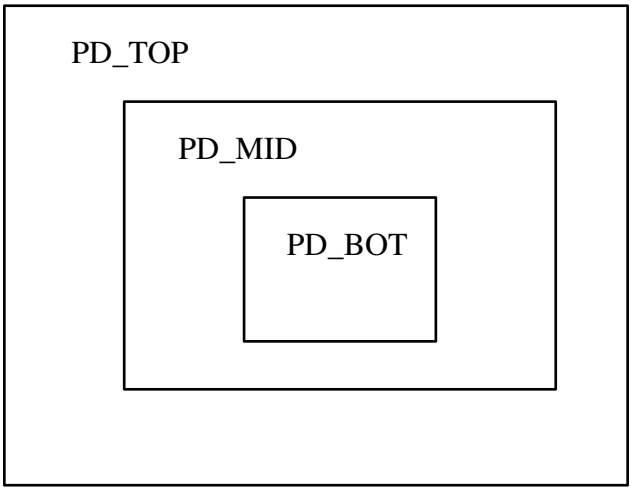
Associating supply sets with an unequal number of functions causes the supply set with the lesser functions to inherit the remaining functions from the associated supply set. The following example first creates a two-function supply set (SS1), then creates a four-function supply set (PD1), and finally promotes the SS1 supply set to a four-function supply set.

```
set_design_attributes -elements {.} -attribute enable_bias TRUE
create_supply_set SS1
create_power_domain PD1
associate_supply_set {SS1 PD1.primary}
```

You can associate supply sets across different scopes. Figure 22 illustrates the three scopes.

Feedback

Figure 22 Associated Supplies Across Three Scopes



The following example associates the supplies across the three scopes in Figure 22:

```
create_power_domain PD_TOP -supply {primary SSTop}
create_power_domain PD_MID -supply {primary SSMid}
create_power_domain PD_BOT -supply {primary SSBot}
associate_supply_set {SSTop mid/SSMid mid/bot/SSBot}
```

Refining Bias Supply Functions Automatically

When you set the `enable_bias` design attribute to the `derived` value, the tool automatically inherits bias functions for every supply set in the scope where `enable_bias` is set to `derived`, based on the power and ground functions. You need not specify the bias functions explicitly and if n-well and p-well functions are the same as power and ground respectively, this setting saves your effort to specify supply sets with all the four functions in the scope where bias is enabled.

```
fc_shell> set_design_attributes –elements {.}
                          -attribute enable_bias derived
```

Or

```
fc_shell> set_design_attributes  -elements {a b}
                          -attribute enable_bias derived
```

Note:

- Setting `enable_bias` to `derived` is analogous to setting it to `true` because, in both cases, the `enable_bias` attribute is set and bias functions are required for all supply sets. Thus, all rules that currently apply for

mixing bias and non-bias blocks with `enable_bias` set to `true` hold for `enable_bias derived` as well.

- The interaction of blocks with `enable_bias derived` and `enable_bias false` is identical to the interaction of blocks with `enable_bias true` and `enable_bias false`.

For n-well only support, that is, when the `mv.upf.enable_nwell_only_support` application option is set to `true`, the tool auto-inherits only the n-well function from the power function because, in this case, the p-well function does not exist.

To save the UPF file with the derived bias functions, set the `mv.upf.save_upf_update_derived_bias` application option to `true`:

```
fc_shell> set_app_options -list
          {mv.upf.save_upf_update_derived_bias true}
```

The value of this application option is `false` by default.

Example 1: No Bias Functions Defined

Consider the following UPF with no bias functions defined for SS1 and SS2:

```
set_design_attributes -elements . -attribute enable_bias derived

create_supply_set SS1  -function {power VDD1}  -function {ground VSS}

create_supply_set SS2  -function {power VDD2}  -function {ground VSS}

commit_upf
```

For this UPF, the `commit_upf` command does not produce any MV-003 error. The auto-derived bias functions for supply sets SS1 and SS2 are:

```
Supply Set              :  SS1
Current Scope           :  leon3 (top level)

Functions               :  -- Name --       -- Supply Net --
                              power           VDD1
                              ground          VSS
                              nwell           VDD1
                              pwell           VSS

Supply Set              :  SS2
Current Scope           :  leon3 (top level)

Functions               :  -- Name --       -- Supply Net --
                              power           VDD2
                              ground          VSS
                              nwell           VDD2
                              pwell           VSS
```

Feedback

Example 2: Partial Bias Functions Defined

When you use `enable_bias derived` at the time of `commit_upf`, the tool auto-inherits bias functions only for supply sets that remain unresolved.

Consider the following UPF where n-well and p-well are explicitly defined for SS1 and not defined for SS2:

```
set_design_attributes -elements . -attribute enable_bias derived

create_supply_set SS1  -function {power VDD1}  -function {ground VSS}
                       -function {nwell VDD2}  -function {pwell VSS}

create_supply_set SS2  -function {power VDD2}  -function {ground VSS}

commit_upf
```

For this UPF too, the `commit_upf` command does not produce any MV-003 error. For SS1, because n-well and p-well are explicitly defined, the tool does not auto-inherit the bias functions. For SS2, the tool auto-inherits the n-well and p-well bias functions.

```
Supply Set             :  SS1
Current Scope          :  leon3 (top level)

Functions              :  -- Name --       -- Supply Net --
                             power           VDD1
                             ground          VSS
                             nwell           VDD2
                             pwell           VSS

Supply Set             :  SS2
Current Scope          :  leon3 (top level)

Functions              :  -- Name --       -- Supply Net --
                             power           VDD2
                             ground          VSS
                             nwell           VDD2
                             pwell           VSS
```

Example 3: N-Well Only Support

Consider the following UPF where the `mv.upf.enable_nwell_only_support` application option is set to `true` for n-well only support:

```
set_design_attributes -elements . -attribute enable_bias derived

create_supply_set SS1  -function {power VDD1}  -function {ground VSS}

set_app_options -list {mv.upf.enable_nwell_only_support true}

commit_upf
```

Feedback

On commit_upf, the tool auto-inherits only the n-well function from the power function.

```
Supply Set              :  SS1
Current Scope           :  leon3 (top level)

Functions               :  -- Name --       -- Supply Net --
                           power            VDD1
                           ground           VSS
                           nwell            VDD1
                           pwell
```

Correlated Grouping of Supply Voltage Triplets

If the voltage variation for each supply is correlated with, not independent of, the other supplies, you can define the supplies as correlated, so that the tool considers only the minimum with minimum voltages, the nominal with nominal voltages, and the maximum with maximum voltages, without mixing between minimum, nominal, and maximum. This method of analysis is called correlated grouping of voltage triplets.

The tool supports correlated grouping of the minimum, nominal, and maximum voltages specified as triplets in the add_power_state command. To define one or more groups of correlated supply nets, use the correlated_supply_group attribute with the set_design_attributes command.

For example, the following command groups SS1 and SS2 supply sets into a correlated supply group and sets the supply voltages as correlated triplets:

```
fc_shell> set_design_attributes -elements {.} \
    -attribute correlated_supply_group "{SS1 SS2}"
```

You can define the power state table as follows:

```
fc_shell> create_supply_set SS1
fc_shell> create_supply_set SS2
fc_shell> add_power_state SS1 -state HV \
    {-supply_expr {power == {FULL_ON 0.9, 1.0, 1.1} && \
    ground =={FULL_ON 0.0}}}
fc_shell> add_power_state SS2 -state HV \
    {-supply_expr {power == {FULL_ON 1.0, 1.1, 1.1} && \
    ground=={FULL_ON 0.0}}}
fc_shell> create_power_state_group PSG
fc_shell> add_power_state -group PSG -state SYSON \
    {-logic_expr {SS1 == HV && SS2 == HV}}
```

The tool analyzes the design behavior with correlated SS1 and SS2 supplies, without mixing between minimum, nominal, and maximum voltages.

For more information about using the set_design_attributes command, see Setting UPF Attributes on Hierarchical Cells.

For more information on power states and power state tables, see Power State Tables.

Querying for Related Supply Sets

To obtain a collection of related supply sets for a given collection of design pins and ports, use the `get_related_supply_set` command. The command takes a collection of existing design pins and ports, or a list of pattern strings to use to generate such a collection, as input. The command returns a collection of related supply set objects in the design. You can use the `-quiet` option to suppress any warnings generated during pin or port lookup.

For an object which is a top-level port or a leaf-level pin and if the related supply set can be derived from the object, you can use the `get_upf_port_attribute` command with the `UPF_related_supply_set` attribute. The value of this attribute for this object will be the resolved supply set.

Comparing Voltage Levels and Voltage Status

Before you define your isolation and level-shifter strategies, you might want to compare the voltage status and voltage levels of two supplies. To compare two supplies, use the `compare_supplies` command. This command compares the first supply (designated as the *reference supply*) to a second supply (designated as the *specified supply*).

The `compare_supplies` command provides the following options:

- `-on_status`

  Compares the specified supply to the reference supply and returns: `equal`, `more_on`, `less_on`, or `independent`.

- `-voltage_level`

  Compares the specified supply to the reference supply and returns: `equal`, `higher`, `lower`, or `independent`.

When the tool compares the voltage levels of two supplies that belong to the same correlated supply group, the following rules apply:

- If the minimum, nominal, and maximum voltages of the specified supply are equal to the corresponding voltages of the reference supply, the command returns `equal`.
- If the minimum, nominal, and maximum voltages of the specified supply are less than the corresponding voltages of the reference supply, the command returns `lower`.
- If the minimum nominal, and maximum voltages of the specified supply are greater than the corresponding voltages of the reference supply, the command returns `higher`.
- If none of the preceding conditions are true, the command returns `independent`.

If you compare the voltages of two supplies that are not part of the same correlated supply group, the following rules apply:

- If the minimum, nominal, and maximum voltages of the two supplies are equal, the command returns `equal`.
- If the minimum voltage of the reference supply is greater than the maximum voltage of the specified supply and rule 1 does not apply, the command returns `lower`.
- If the maximum voltage of the reference supply is less than the minimum voltage of the specified supply and rule 1 does not apply, the command returns `higher`.
- If none of the preceding conditions are true, the command returns `independent`.

Specifying Level-Shifter Strategies

In a multivoltage design, a level-shifter cell is required where a signal crosses from one power domain to another. The `set_level_shifter` and `use_interface_cell` commands allow you to specify the strategy for inserting level-shifter cells.

Topics in this section:

Defining the Level-Shifter Strategy

The `set_level_shifter` command defines a strategy for inserting level-shifter cells between power domains that operate at different voltages. Level shifters are inserted by the tool during the execution of the `compile_fusion` command.

If a voltage violation exists at the domain boundary, the tool inserts level shifters at the domain boundary by default, even when a level-shifter strategy is not defined. This flexibility allows the tool to use the strategy that gets the best results. To restrict the

insertion of level shifters to domain boundaries where the `set_level_shifter` command is used, set the `mv.upf.insert_ls_on_user_cstr_only` application option to `true`.

For additional flexibility in inserting level shifters, use the `-applies_to_boundary` option to specify which power domain boundary to apply the strategy. For more information, see Overview of Power Domain Boundaries.

Specifying a strategy does not force a level-shifter cell to be inserted unconditionally. The tool uses the power state table and the specified rules, such as threshold, to determine where level shifters are needed. When the tool identifies a potential voltage violation, it tries to resolve the violation by inserting multiple level-shifters or a combination of level-shifter and isolation cells.

Specify the elements to which the strategy applies as follows:

- Use the `-elements` to specify a list of ports and pins in the domain to which the strategy applies. Specifying the `-elements {.}` option applies the strategy to all elements in the current scope.
- Use the `-exclude_elements` to exclude a list of ports and pins from the strategy. Specifying the `-exclude_elements {.}` option excludes the strategy from all elements in the current scope.

Using the `-force_shift` option of the `set_level_shifter` command ensures that a level-shifter strategy is applied to the specified elements, irrespective of the default priority order. The level-shifter cells inserted with this option cannot be optimized away by the tool.

To prevent the insertion of level-shifter cells for the specified elements of the power domain, use the `-no_shift` option.

The `-threshold` option specifies the minimum voltage difference between a driver and load that cause level shifter insertion to occur. Using this option overrides any threshold values specified in the cell libraries. The `-rule` option specifies the type of voltage difference to which the `-threshold` option applies; valid values for the -rule option are `low_to_high`, `high_to_low`, or `both`. The tool uses the voltages from the power state definitions to determine these voltage differences.

To specify the source supply set that applies to the elements of the strategy, use the `-source` option. To specify the sink supply set, use the `-sink` option.

The `-input_supply` and `-output_supply` options specify the power and ground supply connections for inserted level-shifter cells based on the driver and load supplies in the path and the available supplies in the domain. The default supply is used when the input or output supply is not specified. If the specified driver and load supplies and an equivalent supply are not available, the tool does not insert any level-shifter cells.

Use the `-update` option to add information to a level shifter strategy. You must use either the `-elements` or `-exclude_elements` option with the `-update` option.

Feedback

The `-name_prefix` and `-name_suffix` options control the naming of the inserted level-shifter cell instances.

Controlling Level-Shifter Cell Locations

For additional flexibility in inserting level shifters, use the `-applies_to_boundary` option of the `set_level_shifter` command to specify which power domain boundary to apply the strategy. For more information, see Overview of Power Domain Boundaries.

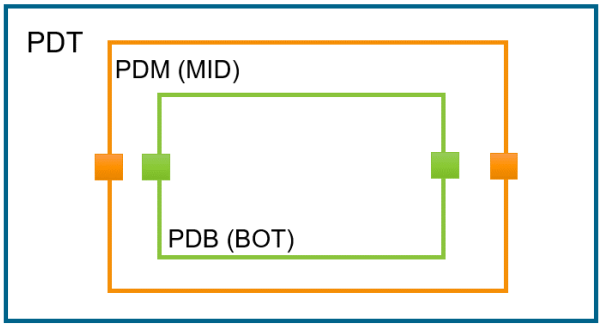
You can use the `-location` option of the `set_level_shifter` command to control the location of level-shifter cells, as follows:

- The `-location automatic` setting (the default) allows the tool to choose the location.
- The `-location self` setting specifies that the level-shifter cell is placed inside the model or cell being shifted.
- The `-location parent` setting specifies that the level-shifter cell is placed in the parent of the model or cell being shifted.
- The `-location other` setting specifies that the level-shifter cell is placed in the parent domain for an upper boundary port or in the child domain for a lower boundary port. The upper boundary port cannot be a port of the design top module and the lower boundary port cannot be a pin of a leaf cell of a hard macro.

If you use the `-location` option, you must ensure that the necessary supplies are available in the specified location. You cannot use the `-location` option with the `-update` option.

For example, consider the power domains in Figure 23.

Figure 23 Nested Power Domains



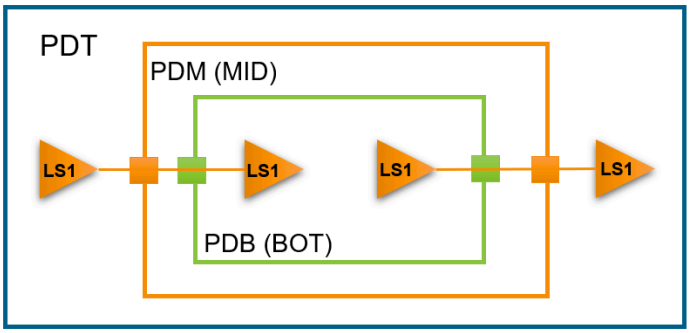
The following UPF commands cause the tool to insert the level-shifter cells shown in Figure 24. Strategy LS1 applies to both the upper and lower boundaries of power domain PDM, for both input and output ports. The tool inserts all level-shifter cells outside the

Feedback

PDM domain because the location is `other`. The tool places the level-shifter cells for the upper boundary ports of domain PDM in the top-level domain and the level-shifter cells for the lower boundary ports of domain PDM in the bottom-level domain.

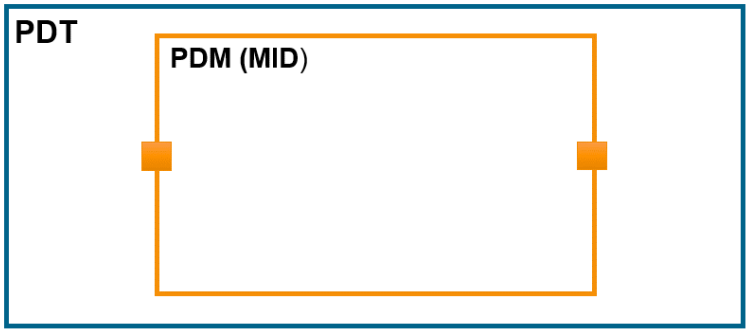
```
create_power_domain PDT
create_power_domain PDM -elements {MID}
create_power_domain PDB -elements {MID/BOT}
set_level_shifter LS1 -domain PDM -applies_to_boundary both \
   -applies_to both -location other
```

Figure 24 Nested Power Domains With Level Shifters



Now consider the simple power domains in Figure 25.

Figure 25 Nested Power Domains

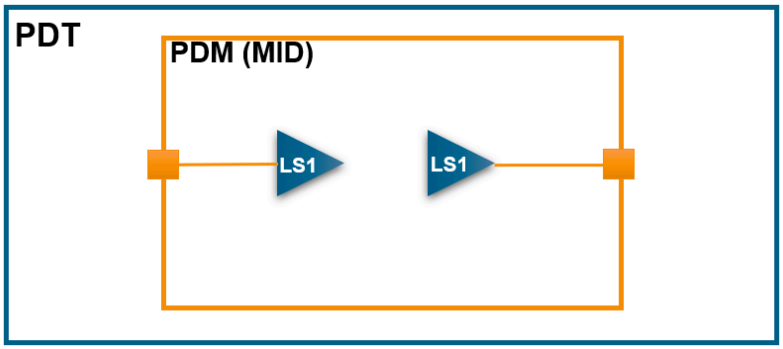


The following UPF commands cause the tool to insert the level-shifter cells shown in Figure 26. Strategy LS1 applies to domain PDT and causes the tool to insert level-shifter cells at the lower boundary of domain PDT, but the location of those cells is inside the PDM domain because the location specifies `other`. Strategy LS2 applies to domain PDM and specifies no level-shifter cell insertion for ports of domain PDM. Therefore the tool does not insert any level-shifter cells for strategy LS2, but this strategy does not prevent implementation of the level-shifter strategy for domain PDT.

Feedback

```
create_power_domain PDT
create_power_domain PDM -elements {MID}
set_level_shifter LS1 -domain PDT -applies_to_boundary lower \
   -applies_to both -location other
set_level_shifter LS2 -domain PDM -applies_to both -no_shift
```

Figure 26 Nested Power Domains With Overlapping Strategies



Resolving Level-Shifter Strategy Precedence

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

1. Strategies applicable to ports, instances, and domains that specify the `-elements` option. Ports have higher precedence than instances, which have higher precedence than domains.
2. Strategies using the `-no_shift` option
3. Strategies using the `-sink` or `-source` options
4. Strategies without any of the previous options

Precedence rules apply to strategies on the same power domain. The `-applies_to` option has no effect on precedence resolution.

If strategies conflict, the tool retains the strategy that was created first and removes those elements from any subsequent strategy definitions. In the following example, strategies LS1 and LS2 are both defined for element p1. To resolve the conflict, the tool applies strategy LS1 to element p1 because strategy LS1 was created first. Strategy LS2 then applies only to element p2.

```
set_level_shifter LS1 -domain PD1 -elements {p1}
set_level_shifter LS2 -domain PD1 -elements {p1 p2}
```

The following examples illustrate level shifter precedence resolution.

Feedback

Example 5 Level-Shifter Precedence Example

```
fc_shell> set_level_shifter LS1 -domain PD1 -sink PD2.primary
fc_shell> set_level_shifter LS2 -domain PD1 -elements block1/p1[0]
```

In Example 5, strategy LS2 has higher precedence because it uses the `-elements` option and applies to a port. The LS1 strategy uses the `-sink` option, which has lower priority.

Example 6 Same Level-Shifter Precedence

```
fc_shell> set_level_shifter LS1 -domain PD1 -elements blk1/p[1]
 -applies_to inputs
fc_shell> set_level_shifter LS2 -domain PD1 -elements blk1/p[1]
```

In Example 6, both strategies have the same precedence. Therefore the tool issues an error message because it cannot determine which strategy to use.

Using Specific Library Cells With Level-Shifter Strategies

When you specify a level-shifter strategy, the tool maps the level-shifter cells to a suitable level-shifter cell in the library. Use the `use_interface_cell` command to limit the set of library cells to be used for the specified level-shifter strategy. This command does not force the insertion of the level-shifter cells. Instead, when the tool inserts the level-shifter cell, it chooses the library cells that are specified with the `-lib_cells` argument of the `use_interface_cell` command. This command has no effect on instantiated level-shifter cells that have a `dont_touch` attribute set on them.

If you specify both the `map_level_shifter_cell` and `use_interface_cell` commands for the same strategy, the `use_interface_cell` command has higher priority.

Allowing Insertion of Level Shifters on Clock Nets and Ideal Nets

By default, the tool does not insert level-shifter cells on clock nets. Set the `mv.upf.auto_ls_clock_nets` application option to specific clock nets, for the tool to insert the level-shifter cells on these nets. Set this variable to "`*`", for the tool to insert level-shifter cells on all clock nets that need level shifters.

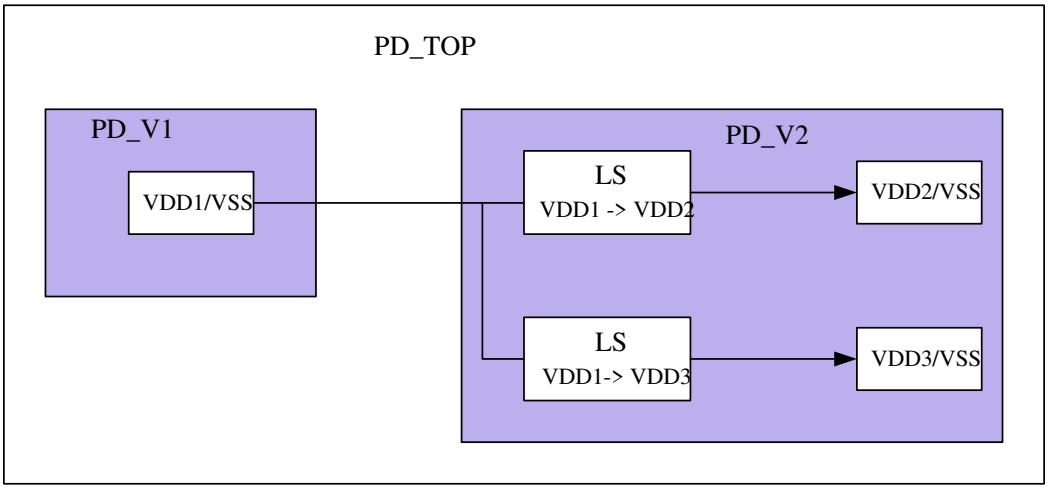
Similarly, the tool does not insert level-shifter cells on ideal nets, by default. Set the `mv.upf.allow_ls_on_ideal_networks` application option to `true` for the tool to insert level-shifters on ideal nets. The default is `false`.

Inserting Multiple Level-Shifter Cells

The Fusion Compiler tool supports multiple level-shifter cell insertion after a boundary pin for flexibility in optimizing designs with heterogeneous fanouts as shown in Figure 27.

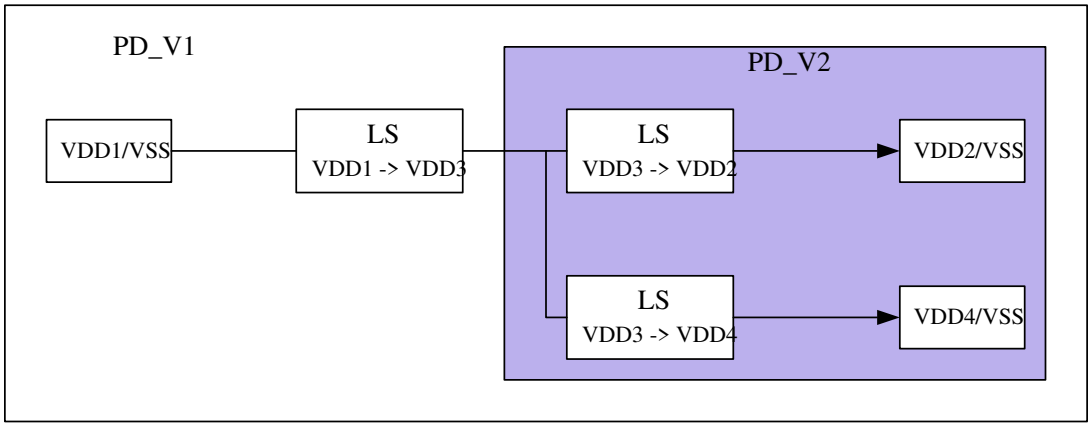
Feedback

Figure 27 Multiple Level Shifter Insertion With Heterogeneous Fanout



More than one level shifter can be inserted at a domain boundary pin as shown in Figure 28. The Fusion Compiler tool can insert one level shifter before the boundary pin and multiple level shifters after the domain boundary pin.

Figure 28 Multiple Level Shifter Insertion at a Boundary Pin

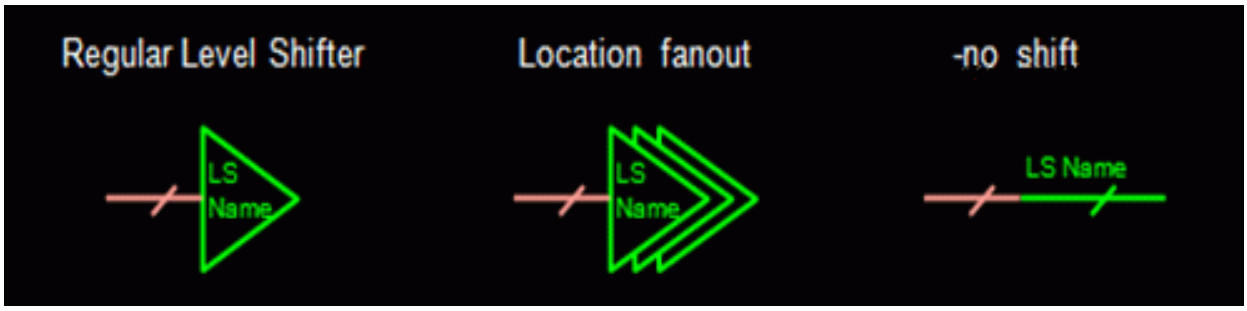


Displaying Level-Shifter Strategies in the GUI

In the Power Domain view, the level-shifter symbol is similar to a buffer symbol and includes a line segment representing the inputs that are shifted, as shown in Figure 29. The location-fanout symbol looks similar to several buffers bundled together and indicates that the level-shifter cells occur on all fanout locations (sink) of the port that they are shifting. The no-shift symbol is represented by a line that shows the continuation of the inputs.

Feedback

Figure 29 Representation of Level-Shifter Strategies

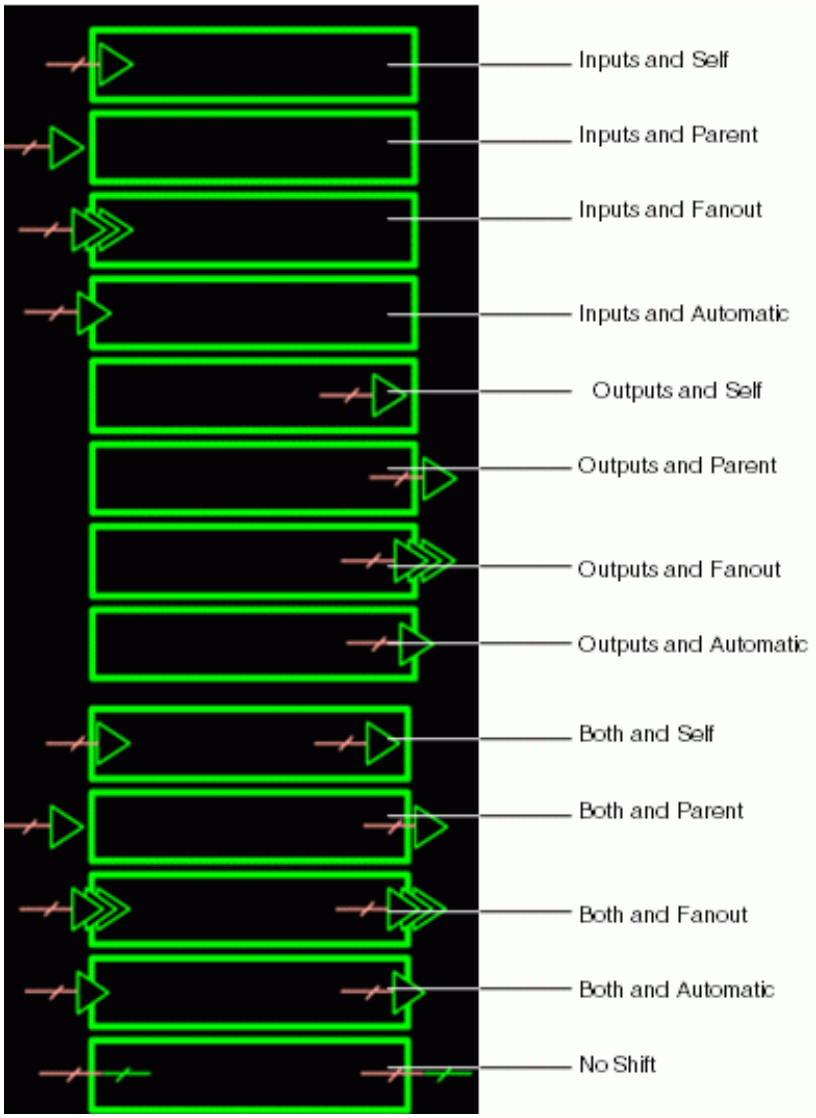


The symbol for each level-shifter strategy is located adjacent to the boundary of its parent power domain. The location depends on whether it shifts inputs or outputs.

Figure 30 shows a domain with level shifters at the input and output of the domain. The symbol appears at the left edge of the boundary if the strategy applies to input ports. The symbol appears to the right edge of the boundary if the strategy applies to the output ports. If the strategy applies to both inputs and outputs, symbols appear at both left and right edges of the boundary.

Feedback

Figure 30 Representation of Level-Shifter Strategies in the Power Domain View



While defining the level-shifter strategy, if you specify the location as `self`, the symbol appears inside the power domain boundary. If you specify the location as `parent`, the symbol appears outside the power domain boundary.

Note:

When you specify a list of elements to the level-shifter strategy using the `set_level_shifter -elements -applies_to` command, the GUI positions the symbol relative to the left or right edge of the power domain boundary, based on whether the list contains input elements, output elements, or both.

Feedback

Understanding Level-Shifter Cell Insertion Errors

If the tool does not insert a level-shifter cell, you can use the `analyze_mv_design -level_shifter` command to obtain more information. If you specify a net or pin with the `-through` option, the report lists the power domains and related supplies for the driver and load sides of the net or pin, along with error messages that indicate why a level-shifter cell was not inserted. The report includes errors about the specific insertion point as well as errors that are applicable to the entire path through the net or pin specified with the `-through` option.

The report is similar to the following:

```
Report : analyze_mv_design
Design : top

-----------------------------------------
Non-default App. Option Info
mv.upf.insert_ls_on_user_cstr_only   true
-----------------------------------------
------------------------------------------------------------------------

Type     Name              Voltage     Power      Related         #Fanout
                           Range       Domain     Supplies
------------------------------------------------------------------------
Driver   TOP_STG1_ISO/Y    0.9-0.9     PD_TOP     vdd_2; vss_1     0
Load     out[4]            1.1-1.1     PD_TOP     vdd_1; vss_1     1
Load     out[5]            1.1-1.1     PD_TOP     vdd_1; vss_1     1
Load     out[6]            1.1-1.1     PD_TOP     vdd_1; vss_1     1
------------------------------------------------------------------------
------------------------------------------------------------------------
Error        Description
------------------------------------------------------------------------
MV-1108      Cannot insert level shifter in the path from driver pin
             'PD_TOP_ISO/Y' (related supply net: vdd_2 [0.9V]) to load
             pin 'out[4]' (related supply net: vdd_1 [1.1V]) because the
             path does not have an effective level shifter strategy.
------------------------------------------------------------------------
------------------------------------------------------------------------
ID  PD Boundary       Const.   PD   Avail  Warnings/          Usable
                                   Supp   Errors             Lib. Cells
------------------------------------------------------------------------
0   in[0] (Driver) -  -        TOP   4                              0
     in[0] (net)
1   in[0] (net) -     -        TOP   4     No errors found          0
    MID/in[0] (Pin)
2   MID/in[0] (Pin)   LS_STG0  MID   2     MV-1104 Cannot fulfill   4
                                           level shifter strategy
                                           due to ...
------------------------------------------------------------------------
```

Feedback

Specifying Isolation Strategies

The `set_isolation` command defines the isolation strategy for a power domain and the elements in the power domain where the strategy is applied. The definition of an isolation strategy contains specification of the enable signal net, the clamp value, and the design elements where the strategy is applied.

The isolation power and ground nets must operate at the same voltage as the primary power and ground nets of the power domain where the isolation cells is located.

Topics in this section:

Defining the Isolation Strategy

The `set_isolation` command specifies the isolation strategy for a power domain. The strategy specifies the elements of the domain to which the strategy applies, the supply set or supply nets to be used for the isolation cells, and any insertion constraints based on the supply sets of the driver and receiver of the net being isolated.

Although the power state table can potentially reduce the number of isolation cells required, isolation synthesis and implementation is entirely based on directives set using the `set_isolation` command. Isolation cell insertion does not consider information in the power state table. If the isolation strategies are not consistent with the allowed states in the power state table, these inconsistencies are reported using the `check_mv_design` command.

Note:

With the `-diff_supply_only` option, you can use the `-source` or the `-sink` option. You cannot use the `-source`, `-sink`, and `-diff_supply_only` options simultaneously.

Where the Strategy Applies

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

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

The `-elements` and `-exclude_elements` options support the `{.}` argument which includes all elements in the current scope.

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

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

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

If you do not specify the `-applies_to` option, the strategy applies to both inputs and outputs of the domain if you use one or both of the `-source` and `-sink` options, or if you set the `-diff_supply_only` option to `true` and the supplies are defined by supply sets rather than supply nets. Otherwise, the strategy applies to outputs only.

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

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

Isolation Constraints Based on Source and Sink Supply Sets

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

- The `-source` option filters the ports connected to a net that is driven by the specified supply set. When you use this option, the supply sets that are associated with each other are considered as connected.
- The `-sink` option filters the ports driving a net that fans out to the logic driven by the specified supply set. Supply sets that are associated with each other are considered as connected. When you specify both the `-source` and `-sink` options, isolation is applied to only those ports that have the specified source and sink.
- The `-diff_supply_only` option determines the isolation behavior between the driver and the receiver supply sets or supply nets. This setting prevents isolation when the driver and receiver cells use matching supplies. When the `-diff_supply_only` option is set to `true`, and the same supply set connects the driver and the receiver of a port on the interface of the reference power domain, the isolation cell is not added in the path from the driver to the receiver.

Isolation Cell Supply Set or Supply Nets

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

- The `-isolation_supply` option of the `set_isolation` command
- The `-supply` (*`supply_set_name`*`default_isolation`) options of the `create_power_domain` command

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

If you do not use the `set_isolation` command option to set the supplies, the `default_isolation` supply set specified by the `-supply` option of the `create_power_domain` command applies.

The Clamp Value

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

Preventing Isolation

In case of conflicting isolation strategies, the tool follows certain precedence rules to determine which strategy to use. For details, see Order of Precedence of Isolation Strategies.

To prevent the insertion of isolation cells for the specified elements of the power domain, use the `-no_isolation` option of the `set_isolation` command. When you use this option, you do not need to specify the isolation power supplies for the strategy.

Isolation Control Signals

The `-isolation_sense` option specifies the logic state of the isolation control signal that places isolation cells in the isolation mode. The possible values for this option are `high` or `low`. The default is `high`.

The isolation signal specified by the `-isolation_signal` option refers to a port, pin, or net with the port or pin having higher precedence. The isolation signal can exist outside the logic hierarchy where the isolation cells are inserted; the synthesis or implementation tools can perform port-punching to make the connection. These punched ports are not considered for isolation or level shifting, even though after port creation, they reside within the coverage of an isolation or level-shifter strategy.

You might have an isolation cell that does not have an enable pin and works as a latch when the input side supply shuts off. To specify this type of cell in the UPF, use the `-isolation_signal {}` option. The UPF should also explicitly indicate whether the cell is a single-rail or dual-rail cell for simulation modeling. Note that it is an error to specify the `-isolation_signal {}` option with the `-isolation_sense` option.

Feedback

Updating the Isolation Strategy

Use the `-update` option to add information to an existing isolation strategy. You can always use this option with either the `-elements` or `-exclude_elements` option. In addition, you can use the `-update` option with the `-location` option in specific situations. For example, you might initially define a strategy without a location, then update the strategy to add a location.

The following requirements apply to the `set_isolation -location -update` command:

- The `set_isolation -location -update` command must occur before the `commit_upf` or `compile` commands.
- You must use the `set_isolation -location -update` command before the `set_isolation_control` command.
- The isolation strategy must have an isolation control signal. However, you cannot use the `set_isolation -location -update` command after the `set_isolation_control` command. Therefore, you must specify the isolation control signal with the `set_isolation` command.
- You cannot use the `set_isolation -location -update` command if the strategy already has a location specified by an earlier `set_isolation -location` command.
- You can use the `set_isolation -location -update` command only one time.

Isolation Cell Naming Conventions

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

Defining Isolation Strategies for DFT Ports

DFT scan-chain stitching might create (punch) ports between power domains. The `set_dft_isolation` command specifies isolation strategies for these ports and updates the UPF with the new information.

You must include the `-ref_strategy` option to specify an existing isolation strategy to apply to DFT ports and the `-ref_domain` option to specify the power domain where the strategy is defined.

The `-dft_target_domain` option allows you to specify different strategies on the reference domain for connections to different load domains. The `-dft_source_domain` option allows you to specify different strategies for connections from different driver domains. These options are not required.

You can optionally apply the isolation strategy only to specific DFT connection types by using the `-type` option. Alternatively, you can exclude connection types by using the `-exclude_type` option.

Feedback

Table 5 lists the supported test signal types.

Table 5 Isolation Connection Types

DFT signal type	Argument of the set_dft_isolation -type option	Description
ScanDataIn	`scan_in`	Scan
ScanDataOut	`scan_out`	Scan
ScanEnable	`scan_enable`	Scan
TestMode	`test_mode`	Scan, clock gating, test controller module (TCM)
ScanClock	`scan_clock`	Scan
LOSPipelineEnable	`los_pipeline_enable`	Scan (pipelined scan enable or PSE)
wrp_clock	`wrp_clock`	Wrapper
wrp_shift	`wrp_shift`	Wrapper
input_wrp_shift	`input_wrp_shift`	Wrapper
output_wrp_shift	`output_wrp_shift`	Wrapper
wrp_ded_capture_in	`wrp_ded_capture_in`	Wrapper
wrp_ded_capture_out	`wrp_ded_capture_out`	Wrapper
wrp_ded_capture_inout	`wrp_ded_capture_inout`	Wrapper
wrp_tcl_test	`wrp_tcl_test`	Wrapper
TestMode	`test_point_clock`	Test point
ScanClock	`test_point_test_mode`	Test mode for test point

For example, the following command applies the iso1 strategy to all DFT connections from the PD1 power domain:

```
fc_shell> set_dft_isolation -ref_strategy iso1 -ref_domain PD1
```

Feedback

The following command applies the iso2 strategy to all ScanDataIn connections from the PD1 power domain to the PD2 power domain:

```
fc_shell> set_dft_isolation -ref_strategy iso2 -ref_domain PD1 \
   -dft_target_domain PD2 -type scan_in
```

If the `set_dft_isolation` command is repeated for a given target domain and connection type, only the first command is honored.

If multiple strategies are specified for the same reference domain, the following precedence rules apply, from highest to lowest priority:

1. A strategy that includes the `-dft_source_domain`, `-dft_target_domain`, and `-type` options
2. A strategy that includes the `-dft_source_domain` and `-dft_target_domain` options, with or without the `-exclude_type` option
3. A strategy that includes the `-dft_target_domain` and `-type` options
4. A strategy that includes the `-dft_target_domain` option, with or without the `-exclude_type` option
5. A strategy that includes the `-dft_source_domain` and `-type` options
6. A strategy that includes the `-dft_source_domain` option, with or without the `-exclude_type` option
7. A strategy that includes the `-type` option
8. A strategy without any options or with only the `-exclude_type` option

Use the `report_dft_isolation` command to report isolation strategies defined by the `set_dft_isolation` command.

Inserting isolation cells at DFT ports might result in redundant isolation insertion. You can reduce or eliminate redundant isolation with the following techniques:

- To specify that the tool should not apply isolation strategies on DFT pins that do not have isolation violations, set the `mv.upf.skip_dft_iso_when_no_violation` application option to `true` (the default is `false`).
- Check whether isolation strategies are applied to both the driver domain and the load domain of DFT paths that have isolation violations.

Feedback

Rules for Location Fanout

Follow these rules while using the `-location fanout` option for the `set_isolation` and `set_isolation_control` commands:

- Do not use the `-isolation_power_net` option of the `set_isolation` command, when you use the `-location fanout`. However, when you use the `-location` option with the value `parent` or `self`, you can use the `-isolation_power_net` option of the `set_isolation` command.
- The `-location fanout` option can be used only when you use one of the `-source`, `-sink`, or `-diff_supply_only` option of the `set_isolation` command. Similarly, when you use the `-elements` and one of `-source`, `-sink`, or `-diff_supply_only` option, you can only specify `fanout` with the `-location` option.
- The `-no_isolation` option cannot be used when you use `-elements` and one of `-source`, `-sink`, or `-diff_supply_only` option of the `set_isolation` command.
- Set the `derived_iso_strategy` attribute, using the `set_design_attributes` command, before specifying `-elements` and one of `-source`, `-sink`, or `-diff_supply_only` option of the `set_isolation` command. After setting the `derived_iso_strategy`, if you do not specify `-elements` and one of `-source`, `-sink`, or `-diff_supply_only` option of the `set_isolation` command, the tool issues an error message.
- If you set the `derived_iso_strategy` attribute, the only value that you can specify with the `-location` option is `fanout`.

For details about setting the UPF attributes on ports and hierarchical cells, see Setting UPF Attributes on Hierarchical Cells.

Order of Precedence of Isolation Strategies

The following isolation strategies have decreasing order of precedence, regardless of the order of execution:

1. Strategies that apply to ports, explicitly specified using the `-elements` option.
2. Strategies that apply to ports, implied by specifying an instance using the `-elements` option.
3. Strategies that apply to ports, implied by specifying only the power domain name.
4. Strategies using the `-no_isolation` option.
5. Strategies using the `-source` and `-sink` options. If both options are used, the strategy takes precedence over any strategy where only one option is used.

6. Strategies where the `-diff_supply_only` option is set to `true`.
7. Strategies where the `-diff_supply_only` option is set to `false`.

If strategies conflict, the tool retains the strategy that was created first and removes those elements from any subsequent strategy definitions. In the following example, strategies ISO1 and ISO2 are both defined for element p1. To resolve the conflict, the tool applies strategy ISO1 to element p1 because strategy ISO1 was created first. Strategy ISO2 then applies only to element p2.

```
set_isolation ISO1 -domain PD1 -elements {p1}
set_isolation ISO2 -domain PD1 -elements {p1 p2}
```

Reporting UPF Strategies for New DFT Ports

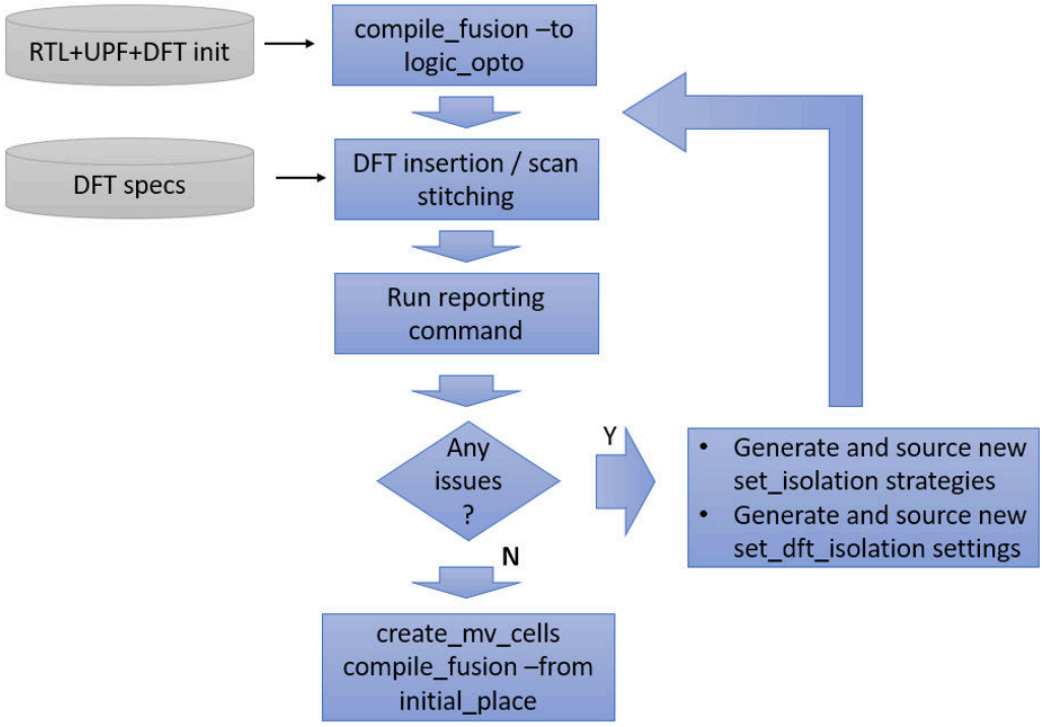
Use the `check_isolation_coverage` command with the `-dft_signals` option for the tool to automatically generate information, to assist you in the selection of isolation policies for DFT signals through the `set_isolation` or `set_dft_isolation` setting.

The `check_isolation_coverage -dft_signals` report includes detailed information on the following:

- Violating DFT paths crossing power domains
- Supply availability
- Available isolation strategies
- Suggested `set_dft_isolation` strategies to fix violating DFT paths

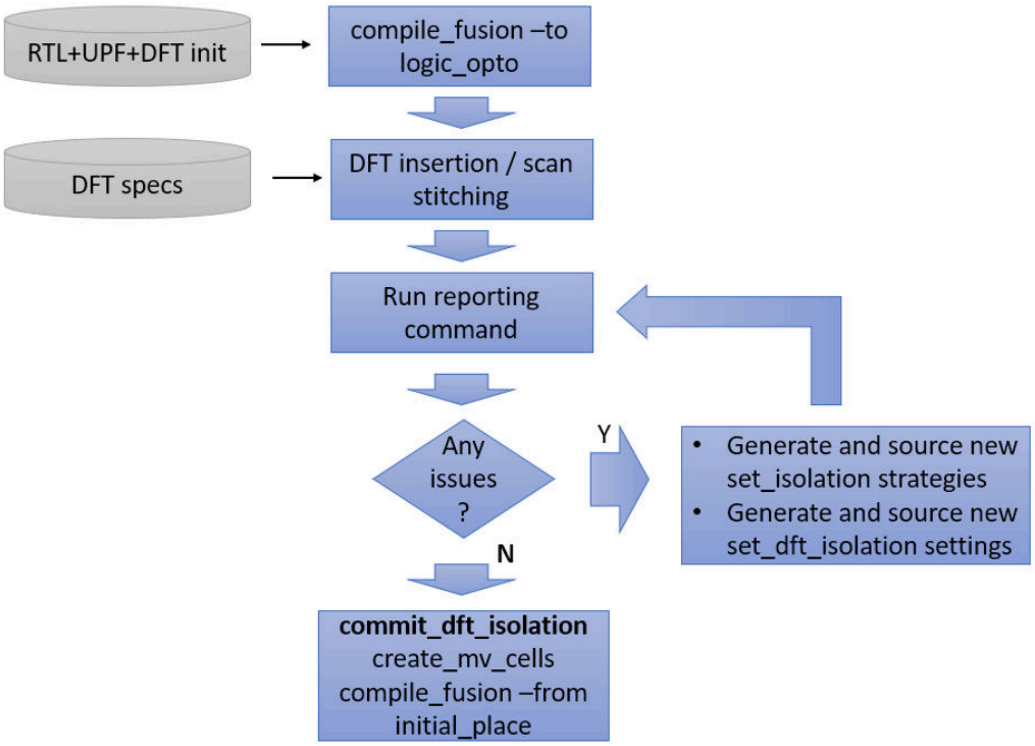
You can use `check_isolation_coverage` in either of the following two ways, to add `set_isolation` or `set_dft_isolation` strategies to fully cover the isolation requirements of DFT signal paths:

1. Add appropriate strategies and rerun `insert_dft`:



Feedback

2. Add incremental `set_dft_isolation` strategies and use `commit_dft_isolation` after all the strategies are specified:



   The `commit_dft_isolation` command commits all existing `set_dft_isolation` settings, applying corresponding isolation strategies to all the affected DFT signals.

The following is a complete list of `check_isolation_coverage` options and their description:

- `-dft_signals`: Limits the report to perform analysis on data paths containing any of the tool-supported DFT signals.

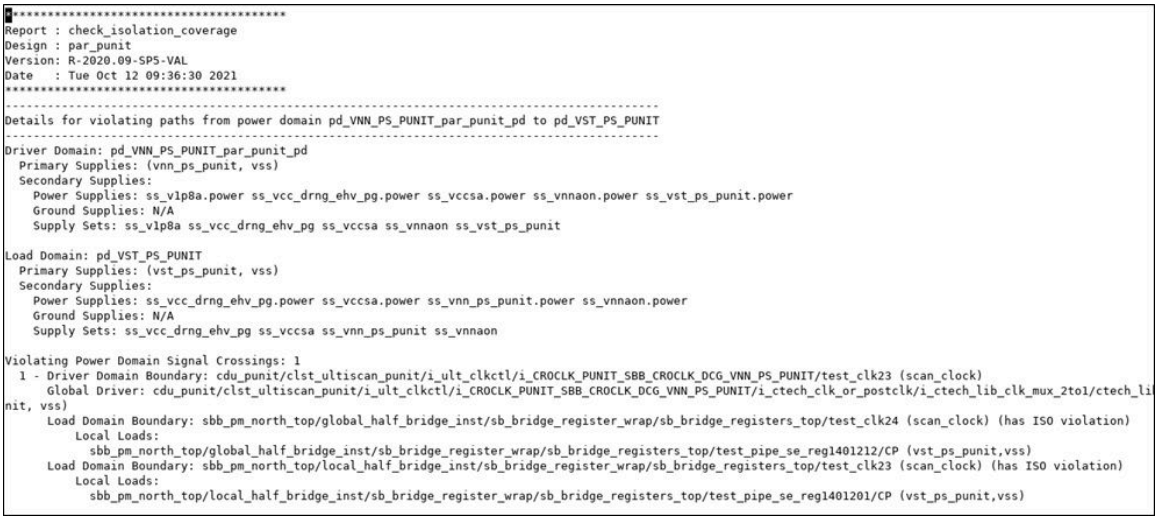
  The supported DFT signal types are: `scan_clock`, `scan_in`, `scan_out`, `scan_in_out`, `scan_enable`, `test_mode`, `input_wrp_shift`, `output_wrp_shift`, `wrp_shift`, `wrp_clock`, `wrp_ded_capture_in`, `wrp_ded_capture_out`, `wrp_ded_capture_inout`, `wrp_td_test`, `wrp_safe_in`, `wrp_safe_out`, `pse_clock`, `los_pipeline_enable`, `test_point_clock`, and `test_point_test_mode`.

- `-from` *`power_domain_name_list`* (Optional): Restricts data path analysis to those where the global driver is located in any of the referenced power domains in *`power_domain_name_list`*.

Feedback

- `-to` *`power_domain_name_list`* (Optional): Restricts data path analysis to those where at least one global load is located in any of the referenced power domains in *`power_domain_name_list`*.
- `-clamp_value` *`clamp_value_list`* (Optional): By default, the tool evaluates isolation strategies regardless of the clamp value. Use this option to restrict the isolation strategy evaluation to those that match the clamp values listed in *`clamp_value_list`*. Allowed values are: 1, 0, “latch” and “all”; “all” means 1, 0 and “latch”
- `-ignore_unimplemented_strategies` (Optional): By default, tool only reports data signal paths on which an isolation violation is found that is not covered by an existing isolation strategy (either implemented or to be implemented). When this option is used, tool checks only for implemented isolation strategies.

Figure 31 check_isolation_coverage Report Example 1



```
****************************************
Report : check_isolation_coverage
Design : par_punit
Version: R-2020.09-SP5-VAL
Date   : Tue Oct 12 09:36:30 2021
****************************************
--------------------------------------------------------------------------------
Details for violating paths from power domain pd_VNN_PS_PUNIT_par_punit_pd to pd_VST_PS_PUNIT
--------------------------------------------------------------------------------
Driver Domain: pd_VNN_PS_PUNIT_par_punit_pd
  Primary Supplies: (vnn_ps_punit, vss)
  Secondary Supplies:
    Power Supplies: ss_v1p8a.power ss_vcc_drng_ehv_pg.power ss_vccsa.power ss_vnnaon.power ss_vst_ps_punit.power
    Ground Supplies: N/A
    Supply Sets: ss_v1p8a ss_vcc_drng_ehv_pg ss_vccsa ss_vnnaon ss_vst_ps_punit

Load Domain: pd_VST_PS_PUNIT
  Primary Supplies: (vst_ps_punit, vss)
  Secondary Supplies:
    Power Supplies: ss_vcc_drng_ehv_pg.power ss_vccsa.power ss_vnn_ps_punit.power ss_vnnaon.power
    Ground Supplies: N/A
    Supply Sets: ss_vcc_drng_ehv_pg ss_vccsa ss_vnn_ps_punit ss_vnnaon

Violating Power Domain Signal Crossings: 1
  1 - Driver Domain Boundary: cdu_punit/clst_ultiscan_punit/i_ult_clkctl/i_CROCLK_PUNIT_SBB_CROCLK_DCG_VNN_PS_PUNIT/test_clk23 (scan_clock)
      Global Driver: cdu_punit/clst_ultiscan_punit/i_ult_clkctl/i_CROCLK_PUNIT_SBB_CROCLK_DCG_VNN_PS_PUNIT/i_ctech_clk_or_postclk/i_ctech_lib_clk_mux_2to1/ctech_li
nit, vss)
      Load Domain Boundary: sbb_pm_north_top/global_half_bridge_inst/sb_bridge_register_wrap/sb_bridge_registers_top/test_clk24 (scan_clock) (has ISO violation)
          Local Loads:
            sbb_pm_north_top/global_half_bridge_inst/sb_bridge_register_wrap/sb_bridge_registers_top/test_pipe_se_reg1401212/CP (vst_ps_punit,vss)
      Load Domain Boundary: sbb_pm_north_top/local_half_bridge_inst/sb_bridge_register_wrap/sb_bridge_registers_top/test_clk23 (scan_clock) (has ISO violation)
          Local Loads:
            sbb_pm_north_top/local_half_bridge_inst/sb_bridge_register_wrap/sb_bridge_registers_top/test_pipe_se_reg1401201/CP (vst_ps_punit,vss)
```

Figure 32 check_isolation_coverage Report Example 2

```
Available Isolation Strategies of Power Domain pd_VNN_PS_PUNIT_par_punit_pd: 10
  1 - Isolation Strategy: clip_top_wrap_clip_top_VNNPS2VCCSA
      Details: -location self -isolation_sense low -isolation_signal clip_top_wrap/clip_top/inf_vnn_ps_iso_Pwrgood -isolation_supply pd_VNN_PS_PUNIT_par_punit_pd.supply_6
        set_dft_isolation -ref_domain pd_VNN_PS_PUNIT_par_punit_pd -ref_strategy clip_top_wrap_clip_top_VNNPS2VCCSA -dft_source_domain pd_VNN_PS_PUNIT_par_punit_pd -dft_target_domain pd_VST_PS_PUNIT
  ---
  2 - Isolation Strategy: clip_top_wrap_clip_top_VNNPS2VNNAON
      Details: -location self -isolation_sense low -isolation_signal clip_top_wrap/clip_top/inf_vnn_ps_iso_Pwrgood -isolation_supply pd_VNN_PS_PUNIT_par_punit_pd.supply_6
        set_dft_isolation -ref_domain pd_VNN_PS_PUNIT_par_punit_pd -ref_strategy clip_top_wrap_clip_top_VNNPS2VNNAON -dft_source_domain pd_VNN_PS_PUNIT_par_punit_pd -dft_target_domain pd_VST_PS_PUNIT
  ---
  3 - Isolation Strategy: dts_aon_wrap_dts_aon_dts_output_isolation
      Details: -location self -isolation_sense low -isolation_signal dts_aon_wrap/dts_aon/i_digpwrgood -isolation_supply pd_VNN_PS_PUNIT_par_punit_pd.supply_6
        set_dft_isolation -ref_domain pd_VNN_PS_PUNIT_par_punit_pd -ref_strategy dts_aon_wrap_dts_aon_dts_output_isolation -dft_source_domain pd_VNN_PS_PUNIT_par_punit_pd -dft_target_domain pd_VST_PS_PUNIT
  ---
  4 - Isolation Strategy: pd_VNN_PS_PUNIT_par_punit_pd_clst_ultiscan_punit_to_drng_isol_buffer_wrap_unspecified_0_iso
      Details: -location self -isolation_sense low -isolation_signal punit_socn_sa_pwrgood_on_vnnaon -isolation_supply pd_VNN_PS_PUNIT_par_punit_pd.supply_6
        set_dft_isolation -ref_domain pd_VNN_PS_PUNIT_par_punit_pd -ref_strategy pd_VNN_PS_PUNIT_par_punit_pd_clst_ultiscan_punit_to_drng_isol_buffer_wrap_unspecified_0_iso -dft_source_domain pd_VNN_PS_PUNIT_par_puni
t_pd -dft_target_domain pd_VST_PS_PUNIT
  ---
  5 - Isolation Strategy: pd_VNN_PS_PUNIT_par_punit_pd_drng_isol_buffer_wrap_to_punit_pcg_sbclk_unspecified_0_iso
      Details: -location self -isolation_sense low -isolation_signal punit_wrap/punit/punit_inf_vnn_gated_powergood -isolation_supply pd_VNN_PS_PUNIT_par_punit_pd.supply_6
        set_dft_isolation -ref_domain pd_VNN_PS_PUNIT_par_punit_pd -ref_strategy pd_VNN_PS_PUNIT_par_punit_pd_drng_isol_buffer_wrap_to_punit_pcg_sbclk_unspecified_0_iso -dft_source_domain pd_VNN_PS_PUNIT_par_punit_pd
 -dft_target_domain pd_VST_PS_PUNIT
  ---
  6 - Isolation Strategy: pd_VNN_PS_PUNIT_par_punit_pd_punit_sbb_croclk_pcg_visa_wrap_to_cdu_dfd_punit_unspecified_0_iso
      Details: -location self -isolation_sense low -isolation_signal punit_wrap/punit/punit_inf_vnn_gated_powergood -isolation_supply pd_VNN_PS_PUNIT_par_punit_pd.supply_6
        set_dft_isolation -ref_domain pd_VNN_PS_PUNIT_par_punit_pd -ref_strategy pd_VNN_PS_PUNIT_par_punit_pd_punit_sbb_croclk_pcg_visa_wrap_to_cdu_dfd_punit_unspecified_0_iso -dft_source_domain pd_VNN_PS_PUNIT_par_p
unit_pd -dft_target_domain pd_VST_PS_PUNIT
  ---
```

Feedback

Automatically Deriving Isolation Strategies for DFT Paths

While specifying UPF isolation strategies for DFT paths, `-source` and `-sink` ISO strategies are specified in the UPF to target the DFT paths generated during `insert_dft`. The issue with this approach is that the `-source` and `-sink` ISO strategies targeting DFT paths can also be applied to functional paths. This can result in redundant isolation cells on functional paths.

Also, with the mixed use of both `-element` and `-source` and `sink` based isolation strategies in the UPF, for the same path, it is possible to introduce redundant isolation cells.

To resolve this issue, that is, to avoid applying the ISO strategy targeting DFT paths on functional paths, you can enable the tool to apply the `-source` and `-sink` ISO strategies only on DFT paths.

To automatically derive ISO strategies for DFT paths, perform the following steps:

1. Specify a placeholder `-source` and `-sink` ISO strategy in the RTL UPF to target DFT paths, that is, strategy with an empty element:

   ```
   set_isolation iso_dft -domain PD_BOT -source SS_TOP -sink SS_BOT
    -elements {} ..
   ```

2. Post `insert_dft`, use the `generate_mv_constraints -dft_isolation` command to auto update the placeholder ISO strategies in a power domain with DFT paths.
3. Apply the updated isolation strategies and run `create_mv_cells` to insert relevant isolation cells.

The complete syntax for `generate_mv_constraints -dft_isolation` is the following:

```
generate_mv_constraints -dft_isolation -apply -output filename
```

Where:

- `-dft_isolation`: Updates source/sink ISO strategies with DFT crossings
- `-apply`: Automatically applies the updated isolation strategies
- `-output` *`filename`*: Saves the updated isolation strategies in the specified file

During `generate_mv_constraints -dft_isolation`, the tool considers the following pins/ports on a domain crossing to update the source/sink ISO strategies:

- Newly punched hierarchical pins during `insert_dft` (You can also query these newly punched DFT pins using the `get_dft_hierarchical_pins` command.)
- Any ports or pins on a domain crossing tagged with the predefined attribute `created_during_dft_eco`
- All ports specified in the `set_dft_signal` command

If an existing ISO strategy is already applicable for a DFT port/pin, the tool skips adding this port/pin while updating the source/sink ISO strategies.

Using Specific Library Cells With Isolation Strategies

When you define an isolation strategy, by default the tool associates the isolation strategy with any suitable isolation cell in the library. When the library does not contain a complete set of isolation cells, you can use some of the basic gates as isolation cells.

To associate a specific set of library cells with the isolation strategy, use the `use_interface_cell` command. The `use_interface_cell` command can also be used to associate standard cells used as isolation cells with the isolation strategy.

When designs contain instantiated isolation cells that are associated with an isolation strategy, the `use_interface_cell` command remaps these library cells to the cells specified with the `-lib_cells` argument of the command. If the instantiated isolation cells have `dont_touch` attribute set on them, the command does not remap these cells. The command has no impact on the instantiated isolation cells that are not, or cannot be associated with an isolation strategy.

If you specify both the `map_isolation_cell` and `use_interface_cell` commands for the same strategy, the `use_interface_cell` command has higher priority.

Using Enable Level-Shifter Cells as Isolation Cells

The Fusion Compiler tool can use an enable level shifter cell as an isolation cell.
If the enable level shifter cell library is specified in the `map_isolation_cell` or `use_interface_cell` commands, the tool checks all libraries specified and uses isolation cells even if the enable level shifter library is specified first. If no `map_isolation_cell` or `use_interface_cell` is specified, the tool first checks for isolation library cells. If no matching isolation library cell is available, then the tool uses an enable level shifter cell, if it is available. If the `map_isolation_cell` or `use_interface_cell` commands only specify enable level shifter libraries, then the tool only uses the enable level shifters as isolation cells even if there are available isolation cells.

The only type of enable level shifter you can use for isolation purposes should be modeled as a level shifter followed by an isolation cell.

Feedback

Isolation Cell Renaming

By default, the tool renames any isolation cells that are inserted by another tool to follow Synopsys naming conventions. The tool maps the new names to the original names so that you can run the `read_def` command, `read_sdc` command, and querying commands without encountering any errors.

To check the name mapping, run the `set_query_rules` command. To keep your original isolation cell name, set the following application option:

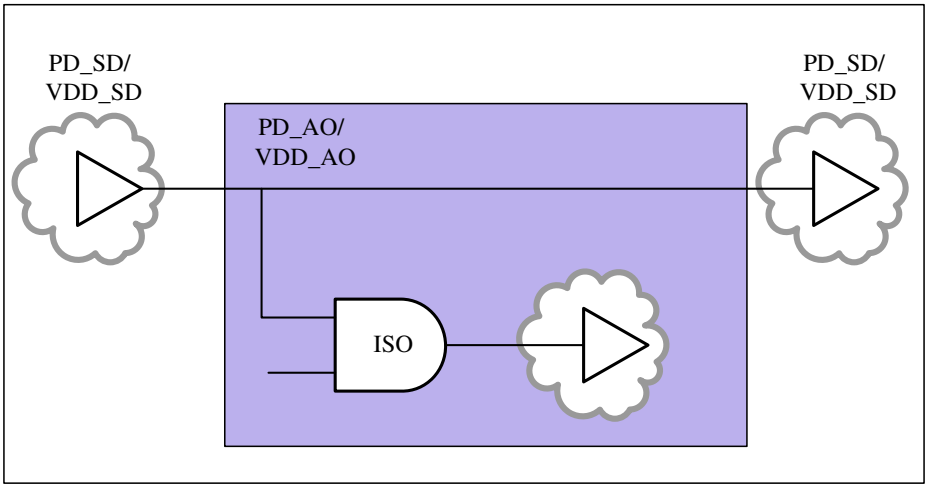
```
fc_shell> set_app_options -name \
   mv.cells.rename_isolation_cell_with_formal_name -value true
```

Isolation Cells and Heterogeneous Loads

The `-source`, `-sink`, and `-diff_supply_only` options for isolation cell insertion allow you to specify an isolation strategy based on the supplies used for the driver and load of a path. It is also useful for paths with loads that have heterogeneous supplies. The Fusion Compiler tool allows you to insert isolation cells using the `-source`, `-sink`, and `-diff_supply_only` options for loads with heterogeneous supplies.

For example, in Figure 33, the isolation cell is inserted on a load with VDD_AO supply and also a load with VDD_SD supply. The `-location` option specifies where the isolation cell is placed.

Figure 33 Isolation Cell With Heterogeneous Loads



For this example, the isolation strategy might be defined as follows:

```
set_isolation S1 -domain PD_AO \
         -sink VDD_AO \
         -diff_supply_only \
         -location self
```

Isolation cell insertion supports name-based association. The tool can optimize the source or sink logic since the association is name-based. A buffer inserted on a path can use a different supply than the isolation cell.

Isolation Cell Insertion on Nets Driven By Constants

A netlist path is driven by a literal constant if the driver has a logic 0 or logic 1 value that is not implemented as a tie cell or direct tie. For example, a logic 0 might be represented in the RTL with the notation 1'b0. The Fusion Compiler tool recognizes literal constants as valid drivers for use with the `-source`, `-sink`, and `-diff_supply_only` options of the `set_isolation` command.

The tool obtains the related supply of the literal constant from a `set_port_attributes -literal_supply` command specified for the constant. If that specification is not available, the tool uses the primary supply of the power domain in which the literal constant is located.

You can prevent the Fusion Compiler tool from inserting isolation cells on constants (literals, such as 1’b0 and 1’b1) that drive macro cell input and primary output ports when the clamp value of the isolation strategy matches the logic value of the constant. For these cases, the tool propagates the constant across the domain boundary and an isolation cell is not needed.

To prevent isolation cell insertion, set the following design attribute:

```
fc_shell> set_design_attributes -attribute \
  relax_constant_corruption_for_macro_inputs_and_primary_outputs \
  true
```

If the clamp value does not match the value of the constant, the tool inserts an isolation cell if a strategy is defined.

Preventing Unnecessary Isolation Cell Insertion

To prevent unnecessary isolation cell insertion, use the `generate_mv_constraints` command after loading the design and the UPF files.

This command generates an isolation strategy with the `-no_isolation` option specified. These strategies are generated for power domain boundary pins where an isolation strategy has been specified, but is not needed based on the power states defined in

Feedback

the UPF files. This command has no effect on domain boundary pins that already have isolation cells inserted. When you use the `-no_isolation` option, you must specify the `-elements` option.

You must specify at least the `-output` or `-apply` option with the `-no_isolation` option. If you do not specify the `-apply` option, you must load the output UPF file manually. For example,

```
fc_shell> generate_mv_constraints -no_isolation -output no_iso.upf
```

The output file no_iso.upf contains the following:

```
# Tool-derived commands
set_isolation snps_no_iso_0 -domain PD1 -no_isolation -elements ...
```

Isolation Handling on Control Signals

Sometimes a port is referenced in a UPF command as the control signal of a UPF strategy or as part of a logic strategy. If the port itself has an isolation strategy, the tool handles isolation cell insertion using certain rules.

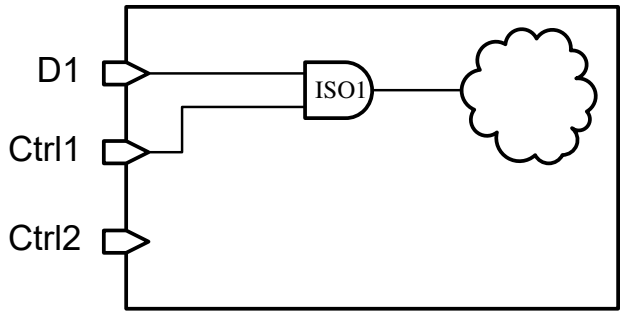
Elements that are not specified as the isolation signal of any isolation strategy are processed first. Then, the tool processes elements specified as the isolation signal of an isolation strategy that is already implemented.

For example, suppose you have the following isolation strategies:

```
set_isolation ISO1 -elements {D1} -isolation_signal Ctrl1
set_isolation ISO2 -elements {Ctrl1} -isolation_signal Ctrl2
set_isolation ISO3 -elements {Ctrl2} -isolation_signal ...
```

The strategy ISO1 has higher priority since its isolation element is not a control signal for any isolation strategy as shown in Figure 34.

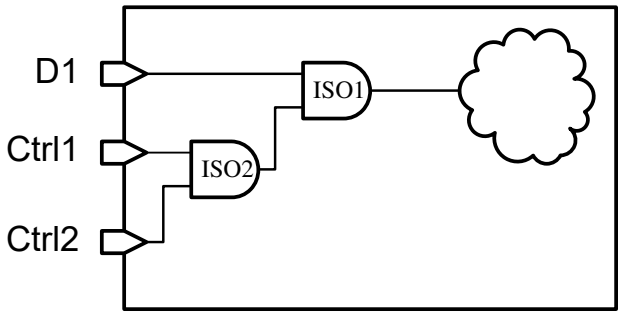
Figure 34 First Step in Isolation Cell Insertion



Isolation strategy ISO2 has the next higher priority since its isolation element, Ctrl1, is already implemented as the control signal of ISO1. This is shown in Figure 35.

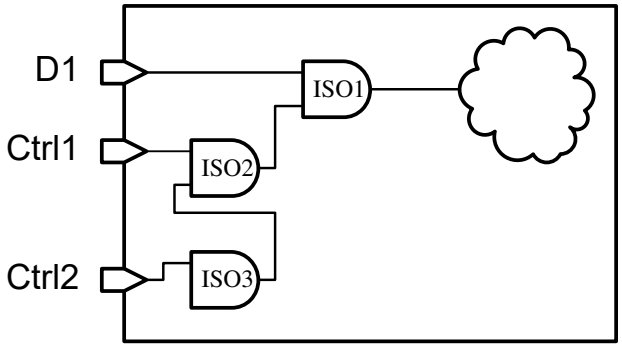
Feedback

Figure 35 Second Step in Isolation Cell Insertion



Isolation strategy, ISO3, can be implemented since the isolation element, Ctrl2, is already implemented as the control signal of ISO. This is shown in Figure 36.

Figure 36 Third Step in Isolation Cell Insertion



Smart Derivation of the -no_isolation Strategy

Currently, the tool always automatically derives the `-no_isolation` strategy on newly punched hierarchical pins for:

- Retention control (save and restore)
- Power switch control
- Isolation control

This is done to ensure that no existing isolation strategy is applied to newly punched ports.

However, this approach has the disadvantage that, if the driver of the control pin is less always-on than the control pin, the isolation violation cannot be fixed with the current isolation strategies, because `-no_isolation` has the highest precedence. There is no way to isolate the newly punched ports on the control path, to add new isolation strategy for them, to fix the isolation violation on the control path.

Feedback

To resolve this issue, you can set the application option `mv.cells.smart_derive_iso_strategy_on_new_control_ports` to true, for the tool to intelligently derive the `-no_isolation` strategy for new punched control pins, based on the PST status. See the following syntax:

```
set_app_options
     -list {mv.cells.smart_derive_iso_strategy_on_new_control_ports true}
```

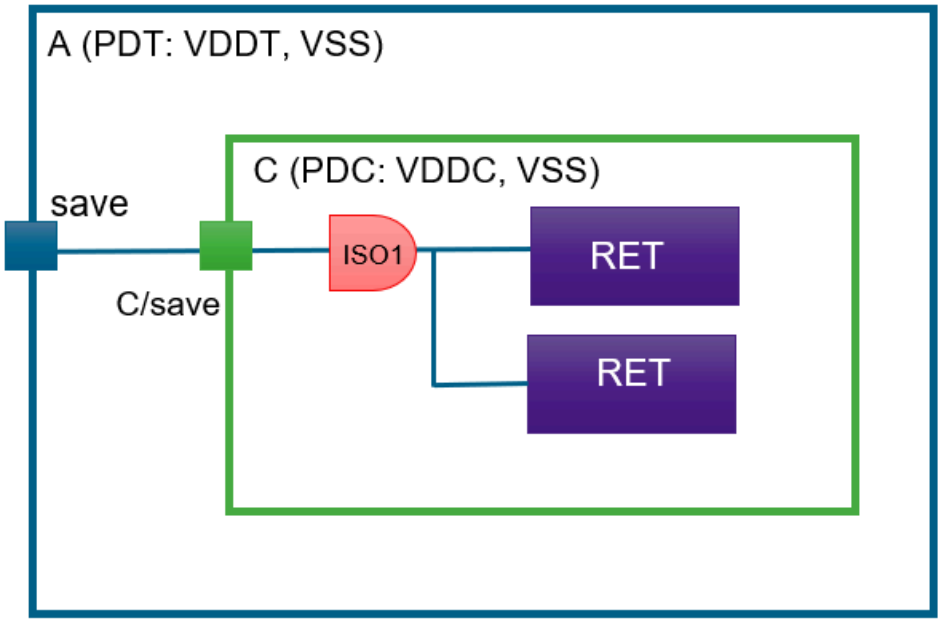
In the smart approach, tool first checks whether the control path has an isolation violation. Only if it finds that there is no isolation violation, the `-no_isolation` strategy is derived for the new ports.

Note:

This feature is currently supported only for new punched pins on the retention control path.

Example

Consider the following example:




```
PST:
       VDDT       |   VDDC
Pst1:  vdd_on     |   vddc_on
Pst2:  vdd_off    |   vddc_on
Pst3:  vdd_on     |   vddc_off

set_retention RET_PDC -domain PDC -elements {C/ret1 C/ret2 …}
  -retention_supply PDC.primary
```

Feedback

```
  -save_signal {save high}  -restore_signal {restore high} …

set_isolation ISO1 -domain PDC -isolation_supply PDC.primary
  -isolation_signal iso1
  -isolation_sense low -location self -clamp_value 0
  -applies_to inputs   -diff_supply_only TRUE
```

After `compile`, the tool punches the new port for save signal C/save. There is an ISO violation on the new port C/save because of Pst2 status. So, with `mv.cells.smart_derive_iso_strategy_on_new_control_ports` set to true, the tool does not derive the `-no_isolation` strategy on pin C/save, applies ISO1 strategy, and inserts the isolation cell on the path of C/save.

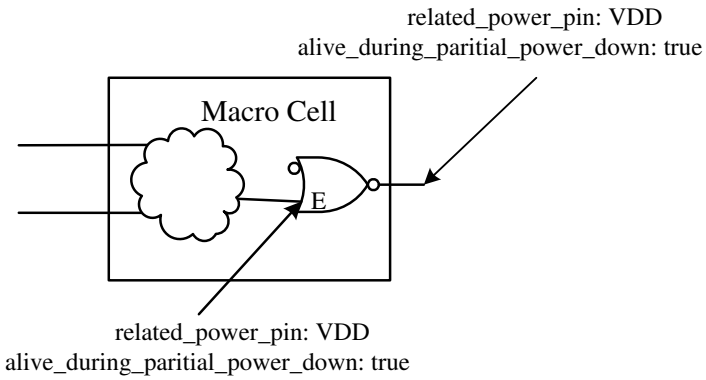
Macro Cells With Internal NOR Isolation Cells

When you model an internally isolated macro cell, a NOR-style isolation cells might appear at the inputs or outputs of the macro cell. If a macro cell contains NOR isolation cells at its output, the Fusion Compiler tool behaves as if the macro has an `is_isolated: true` Liberty attribute specified at the output pin. That is, the tool does not report isolation violations between the NOR-isolated macro output and the logic at the macro's load (assuming the load logic is powered on more than the macro). For details about macro cell modeling, see *The Library Compiler User Guide*.

The tool supports the following macro models when a NOR isolation cell is at the output of a macro:

- The enable pin of the NOR cell is a single pin and has the same `related_power_pin` attribute as the output pin. For example, a macro with a standard cell NOR isolation cell at the output is shown in Figure 37with the specified Liberty attributes.

Figure 37 NOR Cell and Related Macro Cell Attributes



When you use this model, the tool treats the macro cell as if the output pin has an `is_isolated : true` Liberty attribute set. The tool also assumes the following:

- The enable pin and output pin have the same `related_power_pin` attribute value
- The enable pin and output pin have the `alive_during_partial_power_down` attribute set to `true`
- The output pin must have the `isolation_enable_condition` attribute set to a single primary input

- The `isolation_enable_condition` attribute of the macro cell is an equation of all primary inputs to the macro.

  The tool assumes that the output pin has the `alive_during_partial_power_down` attribute set to `true`

- The isolation enable condition is missing, but the `alive_during_partial_power_down` attribute is set to `true`.

  The tool assumes that the output is connected to an ideal always-on supply and does not perform checks on the output pin.

If the `alive_during_power_up` attribute is set to `true` at the macro input pin, the tool performs electrical checks at the macro input pin. If this attribute is `false`, the checks are not performed.

Voltage Checking

To control the voltage checks that the tool performs, use the following application options:

- `mv.upf.nor_iso_macro_allow_enable_supply_check`

  By default, this application option is `true`. If set to `false`, the tool issues a warning message if all the supplies of the enable pins are less on than the sink supply.

- `mv.upf.allow_is_isolated_output_check`

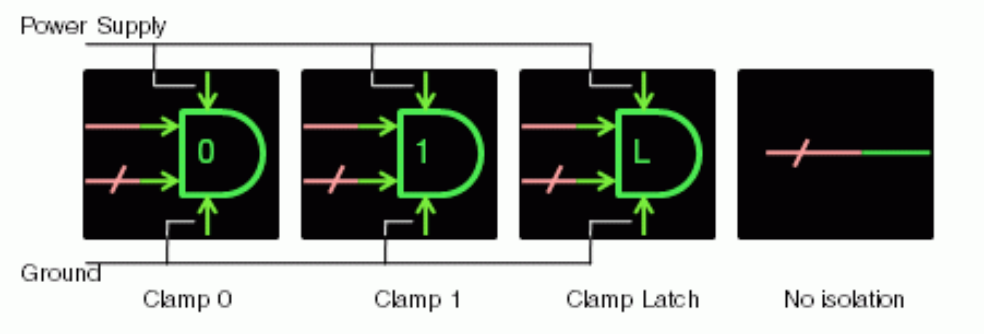
  By default, this application option is `true`. If set to `false`, the tool does not perform voltage checking for isolated output pins with respect to the load supply.

Representing Isolation Strategies in the GUI

Figure 38 shows the symbols used to represent an isolation strategy in the UPF diagram view. The symbol used is similar to an AND gate and the clamp value is shown inside the symbol. The symbol also includes pins for power and ground, a segment representing the isolation signal, and a line segment representing the inputs or outputs that the strategy isolates. When the `-no_isolation` option is specified, a straight line is used to show the continuation of the inputs.

Feedback

Figure 38 Representation of Various Types of Isolation Cells

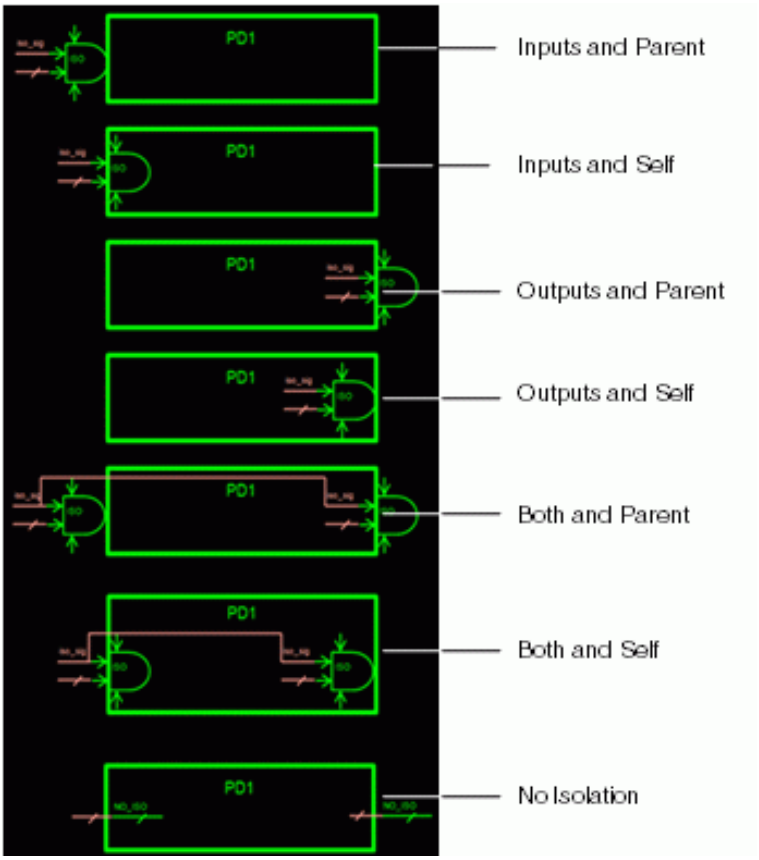


The symbol is located adjacent to the boundary of its parent power domain. The location also depends on whether the strategy isolates inputs or outputs.

Figure 39 shows all possible combinations of isolation strategy symbols and locations, based on the value of the `-applies_to` option of the `set_isolation` command and the value of the `-location` option of the `set_isolation_control` command used in defining isolation strategy.

Feedback

Figure 39 Representation of Isolation Strategies



The symbol appears to the left edge of the power domain boundary if the strategy applies to the input ports. The symbol appears to the right edge of the boundary if the strategy applies to the output ports.

If the strategy applies to both input and output ports, the symbol appears at both left and right edges of the boundary.

While defining the isolation strategy, if you specify the location as `self`, the symbol appears inside the power domain boundary. If you specify the location as `parent`, the symbol appears outside the power domain boundary.

Note:

If you specify a list of elements using the `set_isolation -elements` command, the UPF diagram ignores the `-applies_to` option and positions the isolation symbol relative to the left or right edge of the power domain boundary, based on whether the list contains input elements or output elements or both.

Feedback

Merging and Cloning Multivoltage Cells

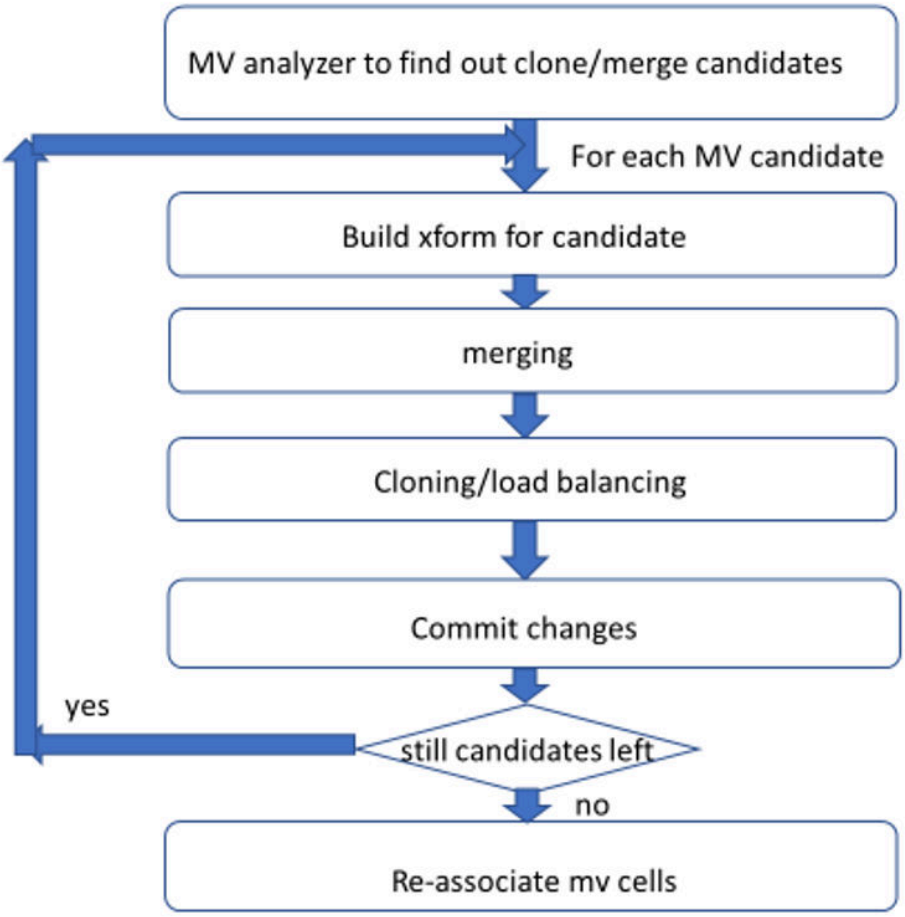
Based on the power strategies specified in the input UPF, the Fusion Compiler tool inserts multivoltage cells, namely, isolation cells, level shifters, and enable level shifters. After multivoltage cell insertion, the tool can perform logic restructuring, that is, merging and cloning of all the isolation, level shifter, and enable level shifter cells, to improve timing and area.

This logic restructuring step is disabled by default. To enable this step, set the `compile.flow.enable_mv_merge_clone` application option to `true`, before running the `compile_fusion` command:

```
fc_shell> set_app_options -name compile.flow.enable_mv_merge_clone
                          -value true
```

The tool executes the multivoltage cell merging and cloning step during the `initial_opto` stage of the `compile_fusion` command.

The following figure illustrates the steps in the multivoltage cell merging and cloning flow:



The tool merges the multivoltage cells if they satisfy all the following conditions:

- The cells have a common local driver.
- The cells logically belong to the same hierarchy.
- The cells are all of the same type, that is, the cells to be merged are all isolation cells, or all level shifter cells, or all enable level shifter cells.
- The cells belong to the same strategy and power domain (for example, isolation or enable level shifter).

Feedback

As a postprocessing step, the tool renames the multivoltage cells that undergo merging and cloning, according to the rules in the following table:

Multivoltage cell	Name of the original (non-merged) cell	Name of the merged cell
Level shifter	*<ls_name_prefix>*_snps_*<power_domain_name>*__*<ls_strategy_name>*_snps_*<pin_name>*_*<inst_index>*_*<ls_name_suffix>*	*<ls_name_prefix>*_snps_*<power_domain_name>*__*<ls_strategy_name>*_snps_*<pin_name>*_***<action>***_*<inst_index>*_*<ls_name_suffix>*
Isolation	*<iso_name_prefix>*_snps_*<power_domain_name>*__*<iso_strategy_name>*_snps_*<pin_name>*_*<inst_index>*_*<iso_name_suffix>*	*<iso_name_prefix>*_snps_*<power_domain_name>*__*<iso_strategy_name>*_snps_*<pin_name>*_***<action>***_*<inst_index>*_*<iso_name_suffix>*
Enable level shifter	*<ls_name_prefix>*_snps_*<ls_power_domain_name>*__*<ls_strategy_name>*_snps_*<iso_name_prefix>*_snps_*<iso_power_domain_name>*__*<iso_strategy_name>*_snps_*<pin_name>*_*<iso_name_suffix>**<ls_name_suffix>*	*<ls_name_prefix>*_snps_*<ls_power_domain_name>*__*<ls_strategy_name>*_snps_*<iso_name_prefix>*_snps_*<iso_power_domain_name>*__*<iso_strategy_name>*_snps_*<pin_name>*_***<action>***_*<iso_name_suffix>**<ls_name_suffix>*

In the preceding table, *action* is either “merge” or “clone”, or “merge_clone”, depending on exactly what action happened last to the specific multivoltage cell.

Note:

“merge_clone” only indicates that both merge and clone happened to the cell; either action can happen first.

As the flow diagram shows, as a last step, after merging and cloning are performed on all the multivoltage cell candidates, the tool runs the `associate_mv_cells` command. This step re-associates all the affected multivoltage cells with the multivoltage strategies and updates all the necessary attributes on the multivoltage cells and the relevant ports. The names of the newly generated merged or cloned cells are unchanged during the re-association.

Limitations

- The Fusion Compiler tool supports the merging and cloning of only isolation, level shifter, and enable level shifter cells. For isolation cells, merging and cloning is not supported for none strategy based NOR-style and NAND-style isolation cells. For example, isolation clamp cell of zero-pin retention cannot be merged or cloned by the tool, since there is no explicit isolation strategy for the clamp cell.

Feedback

- For back-to-back multivoltage cells, merge and clone only works on the multivoltage cells in the load side, if suitable.
- Merge and clone does not work on multivoltage cells on the enable control nets of multivoltage cells.

Specifying Retention Strategies

The `set_retention`, `set_retention_elements`, and `map_retention_cell` commands specify the strategy for inserting retention cells inside shut-down power domains.

Topics in this section:

Defining the Retention Strategy

The `set_retention` and `set_retention_control` commands specify a strategy for inserting retention cells inside a power-down domain.

The `set_retention` command specifies which registers in the power-down domain are to be implemented as retention registers and identifies the save and restore signals for the retention functionality.

The power and ground nets of the retention registers can operate at voltage levels different from the primary and ground supply voltage levels of the power domain where the retention cell is located. Use the `-retention_power_net` and `-retention_ground_net` options to specify the supply nets to be used as the retention power and ground nets. The retention power and ground nets are automatically connected to the implicit save and restore processes and shadow register. If you specify only the `-retention_power_net` option, the primary ground net is used as the retention ground supply. If you specify only the `-retention_ground_net` option, the primary supply net is used as the retention power supply.

The `-retention_supply` option specifies the supply set whose power and ground functions to use as the retention power and retention ground nets.

If specific objects in the power domain do not require retention capabilities, you can specify them with the `-no_retention` option. The tool maps these objects to library cells that do not have retention capability.

The `-save_condition`, `-restore_condition`, and `-retention_condition` options are intended to capture the clock-dependent retention behavior during simulation. The Fusion Compiler tool parses these options, but does not use them. However, the tool preserves the options and writes them out at the `save_upf` command if the netlist is not synthesized.

Every retention strategy defined without the `-no_retention` option must have a corresponding `set_retention_control` command. The `set_retention_control` command specifies the retention control signal and retention sense. The command identifies an existing retention strategy and specifies the save and restore signals and senses for that strategy.

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

The `-assert_r_mutex`, `-assert_s_mutex` , and `-assert_rs_mutex` options of the `set_retention_control` command are intended to capture the clock-dependent retention behavior during simulation. These options are parsed and ignored by the Fusion Compiler tool.

Specifying Elements to Include in the Retention Strategy

The `-elements` option specifies cells for which the retention strategy applies. In the absence of the `-elements` option, the retention strategy is applied to all sequential cells in the power domain, unless you specify the `-no_retention` option. DesignWare instances are supported when using the `-elements` option with the `set_retention` command. The tool applies the `size_only` attribute on all the elements on which it applies the retention strategy.

The `-update` option allows you to refine the element list of a previously defined retention strategy. When used with the `-elements` option, the set of elements is the union of all elements specified for a strategy. You cannot refine a domain-based retention strategy to an element-based retention strategy with the `-update` option. In the following example, the second `set_retention` command results in an error.

Feedback

```
create_power_domain MID -elements {mid1 mid2}
set_retention RET1 -domain MID
set_retention_control RET1 …
map_retention_cell RET1 …
set_retention RET1 -domain MID -elements {mid1} -update
```

The `set_retention_elements` command defines a list of critical elements that can later be used in a `set_retention` command. The list of elements applies to the scope where the `set_retention_elements` is defined. You must retain all of the elements in the list or none of them. It is an error to have a partially retained list of elements.

The `-exclude_elements` option takes the same types of arguments as the `-elements` option. The specified elements must be part of the domain extent.

Use this option to exclude elements from the retention strategy before implementation, such as when you first create a retention strategy with the `set_retention` command. For example, the following command applies the RET1 retention strategy to all registers in the PD1 power domain except for the reg1 register:

```
fc_shell> set_retention RET1 -domain PD1 -exclude_elements {reg1}
```

You can also use the `-exclude_elements` option during successive strategy refinement when you use the `set_retention -update` command before implementation. In the following example, strategy RET1 is applied to register u1/reg1 and strategy RET2 is applied to register u1/reg2 and all other registers in hierarchical cell u1:

```
fc_shell> set_retention RET1 -domain PD1 -elements {u1/reg1 u1/reg2}
fc_shell> set_retention RET2 -domain PD1 -elements {u1}
fc_shell> set_retention RET1 -domain PD1 -exclude_elements {u1/reg2} \
   -update
```

If you write a UPF file before executing an action command, the UPF contains the excluded elements even if they are not in the domain extent. After an action command, the UPF contains only those excluded elements that are in the domain extent.

Resolving Retention Strategy Precedence

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

1. Strategies that apply to registers explicitly specified using the `-elements` option
2. Strategies that apply to registers implied by specifying a Verilog `process` or `always` block
3. Strategies that apply to registers implied by specifying an instance using the `-elements` option
4. Strategies that apply to registers implied by specifying only the power domain name

Feedback

Precedence rules apply to strategies on the same power domain. Retention strategies with the `-no_retention` option have higher precedence than strategies without this option.

If strategies conflict, the tool retains the strategy that was created first and removes those elements from any subsequent strategy definitions. In the following example, strategies RET1 and RET2 are both defined for element p1. To resolve the conflict, the tool applies strategy RET1 to element p1 because strategy RET1 was created first. Strategy RET2 then applies only to element p2.

```
set_retention RET1 -domain PD1 -elements {p1}
set_retention RET2 -domain PD1 -elements {p1 p2}
```

The following examples illustrate retention strategy precedence resolution.

Example 7 General Retention Strategy Resolution

```
set_retention RET1 -domain PD1 -elements {inst1} -no_retention
set_retention RET2 -domain PD1 -elements {inst1/reg1} ...
```

In Example 7, RET1 has the granularity of an instance and RET2 has the granularity of a register. For inst1/reg1, even though RET1 has the `-no_retention` option, it has lower precedence than RET2.

Example 8 Verilog Sample

```
module mid(input clk,d, output reg q1, q2, en);
  always @(posedge clk)
  begin: blk1
    if en ==1'b0
      q1 <= d;
      q3 <= d;
    end
  end
  always @(negedge clk)
  begin: blk2
    q2 <= d;
  end
endmodule
```

In Example 8, the following strategies are defined:

```
set_retention RET1 -domain MID -elements {mid/blk1}
set_retention RET2 -domain MID -elements {mid/q1}
set_retention RET3 -domain MID -elements {mid/blk1}
set_retention RET4 -domain MID -elements {mid/q1}
```

Using the Retention Supply as the Primary Supply

Normally, the primary supply of the domain powers both the retention register and the receiver supply of the retention register's data input. By default, the output driver of the

register is also powered by the primary supply of the domain and therefore, the driver supply of the data output is the primary supply.

However, you can specify that the register and its output is powered by the retention supply by using the `-use_retention_as_primary` option of the `set_retention` command. You cannot update this option. If you use this option, it can affect source and sink analysis during level-shifter, isolation, and repeater cell insertion since the driver or receiver supply uses the retention supply.

When a retention strategy has the `-use_retention_as_primary` option specified, the tool only uses library cells specified in the `map_retention_cell` command that have output pins related to the backup PG pin.

Choosing Specific Library Cells With Retention Strategies

Use the `map_retention_cell` and `map_retention_clamp_cell` commands to constrain the library cell choices for retention registers.

The following guidelines apply to the `map_retention_cell` command:

- The argument of the command must be a retention strategy that is defined for the power domain specified by the `-domain` option. The retention strategy and the `-domain` option are mandatory.
- The optional `-lib_cell_type` option directs the tool to select a retention cell that has the specified cell type. However, the value specified with this option does not change the simulation semantics specified by the `set_retention` command.
- The `retention_cell` attribute on the library cells in the target library defines the retention styles of the library cells.
- The optional `-lib_cells` option specifies a list of library cells that can be used for retention cells.
- The optional `-lib_model_name` option specifies the retention register verification model in the input UPF files, which is parsed for syntax. The model information is primarily used by the Formality tool. The model information is not captured in the design database or in the UPF file written after synthesis. When you specify the port mapping information, the tool accepts either the UPF 2.0 or 2.1 syntax, as follows:

```
map_retention_cell -lib_model_name name1 \
   {-port_map port_name net_ref}*
map_retention_cell -lib_model_name name \
   -port_map {{port_name net_ref}*}
```

The following guidelines apply to the `map_retention_clamp_cell` command:

- The `map_retention_clamp_cell` command is not a UPF command. Therefore it must not be included in UPF files. The tool issues an error message at the `load_upf` command if a UPF file contains the `map_retention_clamp_cell` command. In addition, the tool does not write this command into saved UPF files.
- You can use this command only after loading a UPF file.
- The argument of the command must be a list of retention strategies that are defined for the power domain specified by the `-domain` option. The retention strategies and the `-domain` option are mandatory.
- Use the `-clock_clamp_lib_cells` option to specify the list of library cells that must be used for mapping zero pin retention clamp cells on clock paths.
- Use the `-async_clamp_lib_cells` option to specify the list of library cells that must be used for mapping zero pin retention clamp cells on asynchronous set or reset paths.
- The specified library cells must be isolation or enable level-shifter library cells.

The following guidelines apply to both the `map_retention_cell` and `map_retention_clamp_cell` commands:

- Library cells with `dont_touch` attributes are not used for mapping.
- The operating conditions of the target library cells must match the design environment.
- If none of the specified library cells are suitable, the tool leaves the cells as GTECH isolation cells in the final netlist.
- If a mapping constraint is not specified, the tool selects library cells from the target library in the following order:
  - NOR isolation cells
  - Dual-rail isolation cells
  - GTECH isolation
- If you specify a mapping command more than one time, the tool honors the last successfully accepted command.
- Cell mapping is discarded only if the applicable power domain is removed.

To limit the usage of library cells specified in `map_retention_clamp_cell` for mapping only zero pin retention clamps and no other isolation strategies, set the `mv.upf.iso_map_exclude_zpr_clamp_lib_cells` application option to `true` before running the `map_retention_clamp_cell` command:

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

The default is `false`. However, if you specify the same library cells in both the `map_retention_clamp_cell` and `map_isolation_cell` commands, the `map_isolation_cell` command has higher precedence than the global control provided by the application option.

You can specify the application option with a value `nor` for the exclusion rule to apply to only NOR-type library cells specified in `map_retention_clamp_cell`. In this case, all other types of library cells specified in `map_retention_clamp_cell` can be used for regular isolation mapping.

You can obtain information about retention cell clamp constraints by using the `report_mv_cells -retention_clamp` and `report_power_domains` commands.

Inferring Complex Retention Cells

A complex sequential cell is a cell whose functionality is unknown. You can specify that the tool should infer complex retention cells in place of complex nonretention cells during synthesis. Enable this feature by setting the `mv.cells.infer_complex_retention_cells` application option to `true`.

Usually, the Liberty models of complex sequential cells have pin names that match the pin names of complex retention sequential cells (except for the save and restore pins).

If you have the following in your UPF:

```
set_retention -elements {complex_nonretention_cell}
map_retention_cell -lib_cells {complex_retention_library_cell}
```

The tool tries to infer a complex retention cell during synthesis based on pin name matching. The tool performs the following steps:

1. Matches the pin names of the nonretention cell to the pin names of the retention library cells specified in the `map_retention_cell` command.
2. Chooses the first retention library cell with names that match exactly and uses these in place of the nonretention cells.
3. If no matching retention library cell is found, the tool issues a warning message.

With the pin name matching approach, the tool selects the first matching retention library cell from the `map_retention_cell` command.

If there are multiple matches and if you want to select a particular retention library cell, you should define the `retention_equivalent` attribute first, then set the attribute on the library cells. For example,

```
fc_shell> define_user_attribute -type string -classes lib_cell \
     -name retention_equivalent
fc_shell> set_attribute -objects {nonretention_library_cell} \
```

```
    -name retention_equivalent \
    -value {retention_library_cell}
```

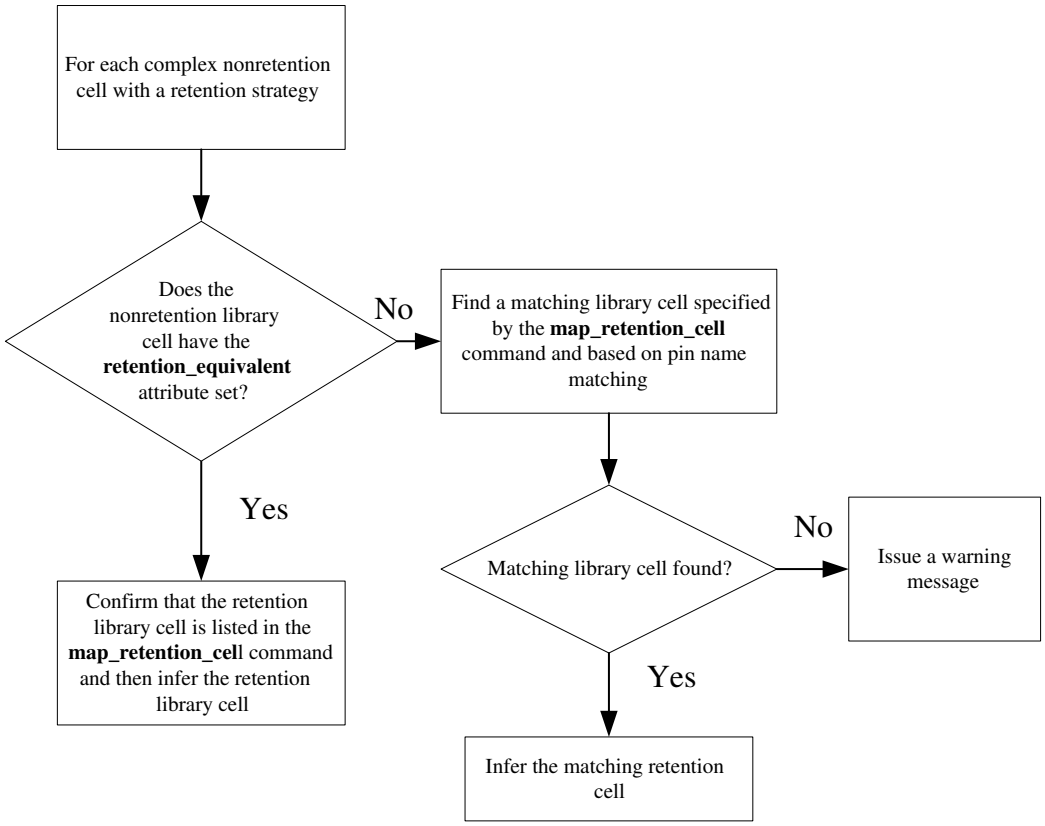
If this attribute is set, a retention library cell can be used in place of a nonretention library cell if the retention library cell is valid. A valid retention library cell has the `retention_cell` attribute, is present in the list of target libraries, and is listed as a library cell in the `map_retention_cell` command. If the library cell is not a valid retention library cell, the tool reverts to pin name matching.

If you have library cells that do not have matching pin names, you can use the `retention_equivalent` attribute to specify the pin mappings. For example,

```
fc_shell> set_attribute -objects {nonretention_library_cell} \
    -name retention_equivalent \
    -value {retention_library_cell {pin1 pin2} {pin3 pin4}}
```

Pin1 of the nonretention library cell corresponds to pin2 of the retention library cell. Pin3 of the nonretention library cell corresponds to pin4 of the retention library cell. If the pin mapping is not valid, then the tool falls back on pin name matching. Figure 40 shows how the tool infers complex retention cells.

Figure 40 Inferring Complex Retention Cells



Feedback

Referencing Verilog Objects as Elements

When using the `-elements` option with the `set_retention` and `set_retention_elements` commands, you can reference Verilog objects as elements. The types of Verilog objects you can reference are:

- HDL blocks
- vector and structure names (`bus` and `struct` instances)
- `generate` block statements

Referencing Verilog HDL Block Names

If you specify an HDL group in the `set_retention` command, the retention strategy applies to all the sequential cells that are members of the specified HDL group, as well as the member cells of the HDL groups that are nested in the specified HDL group. The tool writes out all the leaf-level sequential cells in the element list of the retention commands.

By default, you can reference HDL block names in the UPF. To enable hierarchical naming for HDL block names, set the `hdlin.naming.hierarchical` application option to `true`.

For example, if you have the following Verilog RTL:

```
module top (a, clk, q, gate);
input clk, gate;
input [1:0]a;
output [1:0]q;
reg [1:0]q;
always @ (posedge clk) begin : block_1
  reg[1:0]tmp;
  if (gate)
    q = tmp;
  else
    tmp = a;
end
endmodule
```

The input UPF can use the block_1 name as follows:

```
set_retention ret1 -domain TOP -elements {block_1}
```

If the `hdlin.naming.hierarchical` application option is set to `true`, the output UPF is written as follows:

```
set_retention ret1 -domain TOP -elements {block_1.tmp_reg[0] \
                                          block_1.tmp_reg[1] \
                                          q_reg[0] q_reg{1]}
```

If the `hdlin.naming.hierarchical` application option is set to `false`, the output UPF is written as follows:

```
set_retention ret1 -domain TOP -elements {tmp_reg[0] tmp_reg[1] \
                                          q_reg[0] q_reg[1]}
```

Referencing Verilog Bus and Struct Names as Elements

When you refer to the name of a Verilog `bus` or `struct` in the elements list of the `set_retention` command, the tool writes out all the sequential cells inside the `bus` or `struct` statement. This feature is enabled by default.

For example, consider the following Verilog bus definition:

```
module top (a, clk, q);
input [3:0]a, clk;
output [3:0]q;
reg[3:0]q;
always @ (posedge clk) begin
  q = a;
end
endmodule
```

If the input UPF refers to the q register as follows:

```
set_retention ret1 -domain Top -elements {q}
```

Then the output UPF is written as follows:

```
set_retention ret1 -domain Top -elements {q_reg[3] \
                                          q_reg[2] \
                                          q_reg[1] \
                                          q_reg[0]}
```

Similarly, consider the following Verilog `struct` definition:

```
module top (b, l, r, clk, out)
typedef struct packed {
  bit b;
  logic l;
  reg r;
} ST_0;
input clk, b, l, r;
output ST_0 out;
always @ (posedge clk) begin
  out.b <= b;
  out.l <= l;
  out.r <= r;
end
endmodule
```

If the input UPF refers to the `struct` out as follows:

```
set_retention ret1 -domain Top -elements s{out}
```

The output UPF is as follows:

```
set_retention ret1 -domain Top -elements {out_reg.b \
                                          out_reg.l \
                                          out_reg.r}
```

Referencing Cells Inside a Verilog Generate Block Statement as Elements

When you use the `set_retention` command and you refer to cells inside a Verilog `generate` block statement, the retention strategy applies to the specified cells in the `generate` block. To enable `generate` block statement support, set the `hdlin.naming.upf_compatible` application option to `true`.

When you enable the application option, the tool does the following:

- Sets the `hdlin.naming.hierarchical` application option to `true`
- Sets the `hdlin.naming.structs_records` application option to `"%s.%s"`

Example 9 Generate Block RTL

```
module generate_1(datain, clk, rst, qout, save, restore);
input datain, clk, rst, save, restore;
output [1:0]qout;
reg {1.0}qout;
genvar i;
generate for (i=0; i<2; i=i+1) begin : mygenblk
  always @(posedge clk or negedge rst) being : myblk
    reg myreg;
    if (!rst) begin
      qout[i] = myreg;
      myreg = datain;
    end
    else begin
      qout[i] = myreg;
      myreg = data in;
    end
  end
endgenerate
endmodule
```

To refer to a cell inside a generate statement, specify the `generate` block name with the index followed by the cell name. For example, if your input UPF has the following:

```
set_retention ret1 -domain Top -elements {mygenblk[0].myblk.myreg_reg}
```

The output UPF is as follows:

```
set_retention ret1 -domain Top -elements {mygenblk[0].myblk.myreg_reg}
```

In this example, there is no change in the output UPF since the input UPF specifies a leaf-level cell.

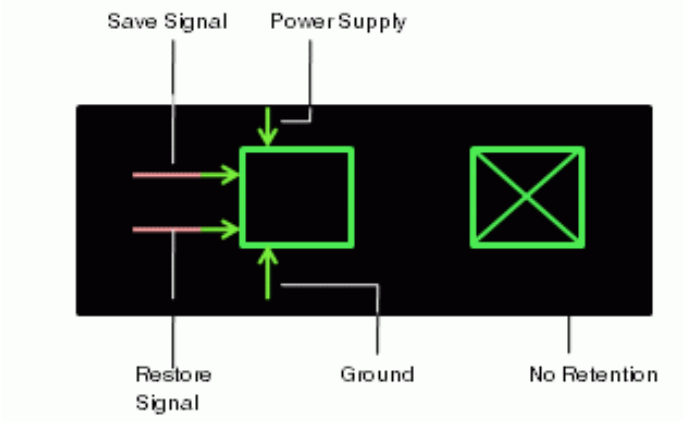
Note:

If a `generate` block is not given a name, the tool creates a default name of the format genblk%d where "%d" represents a nonnegative integer number. If the `generate` block was not named in Example 9, the tool would use `genblk1[1].myblk.myreg_reg` and `genblk1[0].myblk.myreg_reg`.

Representing Retention Strategies in the GUI

In the Multivoltage Power Domain view, the retention cell is represented by a green bounding box as shown in Figure 41. The symbol includes pins for power and ground and segments for save and restore signals. The no-retention symbol contains a “X” inside the bounding box.

Figure 41 Representation of Retention Cells



All retention symbols are located at the center of their parent power domains. The diagram displays the supply nets connected to the retention strategy, the domains to which the strategy belongs and their save and restore signals.

Specifying Repeater Strategies

Repeaters are buffers inserted at regular intervals along the length of a long net to maintain sufficient drive strength along the full length of the net. In the Fusion Compiler tool, the `set_repeater` command defines a strategy for inserting repeater cells (buffers) on the interface of a power domain. The tool inserts a buffer using a specified power supply.

Feedback

Figure 42 Example Using the set_repeater Command

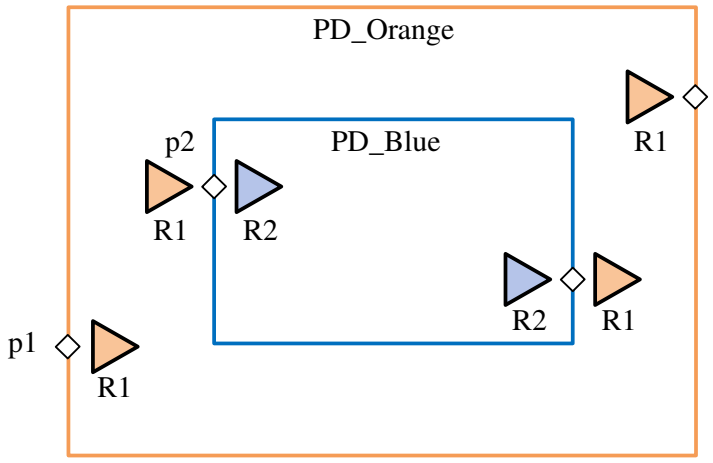


Figure 42 shows an example using the `set_repeater` command. The corresponding script is shown as follows:

```
set_design_attributes -elements {.} \
   -attribute lower_domain_boundary true
set_repeater R1 -domain PD_Orange \
   -repeater_supply ss_orange
set_repeater R2 -domain PD_Blue \
   -elements {.} \
   -repeater_supply ss_blue

set_repeater R1 -domain PD_Blue -repeater_supply SS_Blue
set_isolation ISO1 -domain PD_Blue -isolation_supply SS_Orange \
   -applies_to inputs -sink SS_Blue
set_isolation ISO2 -domain PD_Blue -isolation_supply SS_Blue \
   -applies_to outputs -source SS_Blue
```

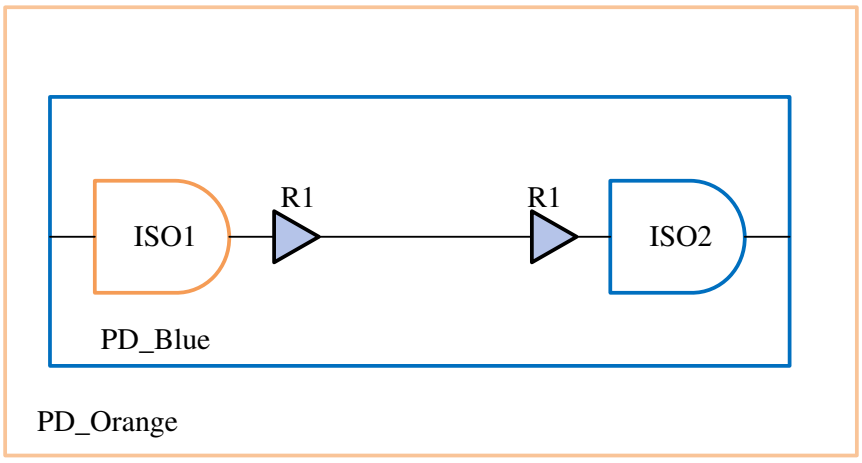
Effect on Power Management Cell Insertion

Because repeater cells are inserted before isolation and level-shifter cells are inserted, the presence of repeaters can affect the implementation of isolation and level-shifter strategies that use the `-source` or `-sink` options. For strategies that use the `-source` and `-sink` options, repeaters are considered endpoints for source and sink analysis.

Isolation and level-shifter cells are placed close to the domain boundary, that is, between the domain crossing and the repeater as shown in Figure 43.

Feedback

Figure 43 Repeaters and Power Management Cells



For the example in Figure 43, the following isolation strategies apply:

```
set_repeater R1 -domain PD_Blue -repeater_supply SS_Blue
set_isolation ISO1 -domain PD_Blue -isolation_supply SS_Orange \
    -applies_to inputs -sink SS_Blue
set_isolation ISO2 -domain PD_Blue -isolation_supply SS_Blue \
    -applies_to outputs -source SS_Blue
```

Creating Power Switches

The `create_power_switch` command creates a virtual instance of a power switch in the scope of the specified power domain. A power switch has at least one input supply port and one output supply port. When the switch is off, the output supply port is shut down and has no power.

The `create_power_switch` command lets the tool know that a generic power switch resides in the design at a specific scope or level of hierarchy. The off state of the power switch output is used in the power state table.

The `map_power_switch` command defines which library cells to use for a specific UPF power switch.

The `-domain` option is available, but not required, for both the `create_power_switch` and `map_power_switch` commands. If you use the `-domain` option, the command scope is the scope of the specified power domain. If you do not use the `-domain` option, the command scope is the current scope and the implicit power domain is the domain of the current scope.

The following example is a definition of a power switch named SW1.

```
create_power_switch SW1 \
  -domain PD_TOP \
```

Feedback

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

The UPF standard requires a simple name for the power switch in a `create_power_switch` command. By default, the tool checks this requirement. To allow the use of hierarchical names, set the `mv.upf.input_enforce_simple_names` application option to `false`.

You can override the location of the power switch by using the `contains_switches` attribute of the `set_design_attributes` command. For the following example, even though you are in scope U0, the power switch is placed in scope U2.

```
set scope U0
create_power_switch SW1 ...
set_design_attributes -elements {U2} -attribute contains_switches {S1}
```

You can use the `contains_switches` attribute more than one time on a hierarchical cell. If you do, the power switches are added to the list of switches related to a hierarchical cell. For example,

```
set_design_attributes -elements {U2} -attribute contains_switches {S1}
set_design-attributes -elements {U2} -attribute contains_switches {S2}
```

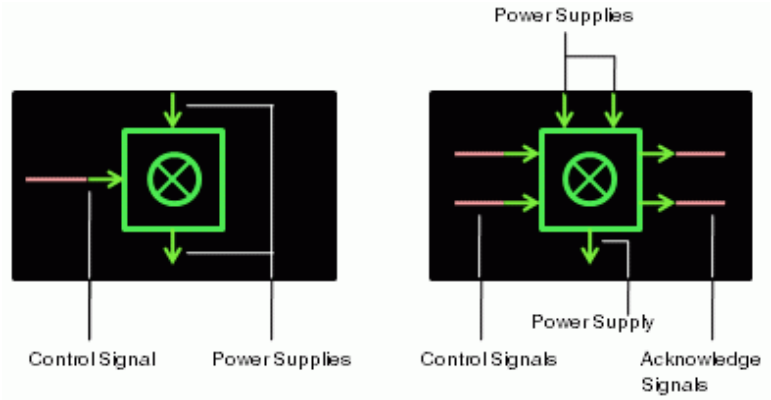
is the same as specifying

```
set_design_attributes -elements {U2} -attribute contains_switches {S1 S2}
```

Power Switches in the UPF Diagram View

In the UPF diagram view, a power switch is represented by a green circle with an X inside it, as shown in Figure 44.

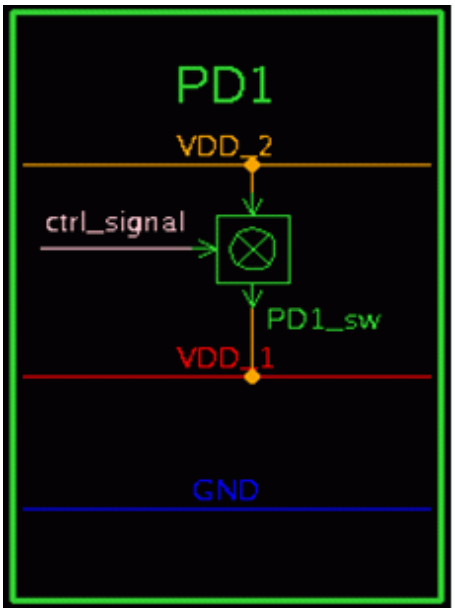
Figure 44 Representation of a Power Switch



The symbol indicates the input and output supply ports, the control ports, and the control signals. The arrows represent the direction of the ports.

As shown in Figure 44, a power switch can have single or multiple control signals. The power switches are located within the boundaries of their parent power domain. Because power switches have supply nets as input and output, they are located between the power supply nets as shown in Figure 45.

Figure 45 Location of the Power Switches in the Power Domain



Power State Tables

A power state table defines the legal combination of states that can exist simultaneously during the operation of the design. A power state table is a set of power states of a design in which each power state is represented as an assignment of power states to individual power nets.

A power state table of a design captures all the possible operational modes of the design in terms of power supply levels. Given a power state, a relationship (including voltages and relative always-on relations) can be inferred between any two power nets.

The Fusion Compiler tool uses the power state table and the specified rules, such as threshold, to determine where level shifters are needed. When the tool identifies a potential voltage violation, it tries to resolve the violation by inserting multiple level shifters or a combination of level-shifter and isolation cells to resolve the violation.

Feedback

Topics in this section:

Defining Power States

Use the `add_power_state` command to define one power state for a supply set. For each power state of the supply set, you must use one `add_power_state` command. By default, undefined power states are considered illegal states.

The tool accepts the following syntax for the `add_power_state` command:

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

The tool also accepts the UPF 2.0 version of the command as follows:

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

In the UPF 2.0 format, the `state_name` value can be specified either inside or outside the curly braces.

You can mix different versions of the `add_power_state` command. However, you must use the same style for the same objects. For example, if you specify one state for the SS1 supply set using UPF 2.1 style, you can use only the UPF 2.1 style for specifying the states for SS1. For example, the following is not allowed:

```
add_power_state SS1 -state S1 {-supply_expr {power == {OFF}}}
add_power_state SS1 -state {S2 -supply_expr {ground == {OFF}}}
```

Use the `-state` option to specify the name of the power state of the supply set.

Feedback

Use the `-supply_expr` option to specify the power state and the voltage value for the supply net components of the supply set.

The UPF standard requires a simple name for the *object_name* argument.

The implementation tools do not use the information specified using the `-simstate` option. The tool reads this option and preserves them in the output UPF. The tool accepts all seven possible simulation states, as specified in the IEEE 1801 language reference manual.

Topics in this section:

- Default Power States
- Power State Propagation
- Specifying Supply Expressions
- Specifying Logic Expressions
- Successive Refinement of Power States

Default Power States

The Fusion Compiler tool supports the use of the default or predefined power states ON and OFF. You can refer to these states during the early definition stage of the UPF, before the actual supply voltages are defined.

You can use the default power states to define other power states in the `add_power_state` command. Using the `-update` option the first time you refer to the default power states is optional, even though the states already exist. However, the `-update` option is required for all subsequent commands that refer to the default states.

You can choose to use the ON and OFF names for other power state definitions.

The following example uses the default state ON in the definition of the new state NOR.

```
create_power_domain TOP -elements {.} -supply {primary}
add_power_state -domain TOP \
   -state {NOR -logic_expr {TOP.primary==ON && ... }
```

You must fully define the default ON and OFF states before performing any action commands or checking commands. For example:

```
add_power_state TOP.primary \
   -state {ON -supply_expr {power=={FULL_ON 1.08} \
   && ground=={FULL_ON 0.0}} -update
```

Feedback

Power State Propagation

You can control the propagation of power states by setting the `enable_state_propagation_in_add_power_state design` attribute on a top-level or block-level design. The default of this attribute is `false`.

When the attribute is `true`, the following conditions apply:

- The tool propagates the name of the supply set state specified in the `add_power_state` command to the functional nets.
- You can subsequently use the net and the supply set state in the `create_pst` and `add_pst_state` commands.
- You cannot use the `&&` operator in the `-supply_expr` option of the `add_power_state` command.

When the attribute is `false`, the following conditions apply:

- The tool does not propagate the name of the supply set state specified in the `add_power_state` command to the functional nets.
- You cannot use the net and the supply set state in the `create_pst` and `add_pst_state` commands. To create power state tables, you must use the `create_power_state_group` command and refer to the supply set state names in the group definition.
- You can use the `&&` and the `||` operators in the `-supply_expr` option of the `add_power_state` command.

You can set the `enable_state_propagation_in_add_power_state` design attribute to any combination of `true` and `false` for top-level or block-level designs.

Specifying Supply Expressions

The supply expression specified with the `-supply_expr` option is used to determine the legal states of the supply nets of the supply set during the synthesis of the design. The supply expression uses the following syntax:

(net == *netstate* || net == *netstate*) && net == *netstate*

The net can be `power` or `ground`, and the *netstate* syntax must be one of the following:

- {*status*}
- {*status nom*}
- {*status min nom max*}

Feedback

Valid values for the status are `OFF` and `FULL_ON`.

The *min*, *nom*, and *max* values are floating point numbers representing the minimum, nominal, and maximum voltages of the specified state.

If the status is `FULL_ON`, you can specify zero to three voltage values. You can defer specifying a voltage value in the initial command, then specify it later using the `-update` option, as shown in the following example:

```
add_power_state -supply SS2 -state ON1 \
     {-supply_expr {power =={FULL_ON} && ground == {FULL_ON}}}
...
add_power_state -supply SS2 -state ON1 \
     {-supply_expr {power == {FULL_ON 1.0} && ground == {FULL_ON 0.0}}} \
      -update
```

The voltage values that you specify with a power state are interpreted as follows:

- When you specify no voltage value, the tool assumes that the value is specified at a later time with the `add_power_state -update` command.
- When you specify a single voltage value, this value is the nominal voltage of the associated state.
- When you specify two voltage values, the first value is the minimum voltage and the second value is the maximum voltage. The average of the two values is considered to be the nominal voltage of the power state.
- When you specify three voltage values, the first value is the minimum voltage, the second value is the nominal voltage, and the third value is the maximum voltage of the power state.
- If you specify more than one voltage value, you must list them in increasing order.

The following example shows the use of the `add_power_state` command to define two power states of the PD1_primary supply set:

```
add_power_state PD1_primary -state HVp \
        { -supply_expr {power == {FULL_ON, 1.08, 2.05, 3.0}}}
add_power_state PD1_primary -state HVg \
        { -supply_expr {ground == {FULL_ON, 0.0}}}
```

The tool supports parentheses in the Boolean supply expression. For example,

```
add_power_state SS_AO -state {ON -supply_expr {(power == {FULL_ON 1.0}) \
        && (ground == {FULL_ON 0.0})}}
```

Using the OR Operator in add_power_state -supply_expr

When `add_power_state` reads a `-supply_expr` expression with combinations of ||, &&, and == operators, the tool will resolve the command with `-supply_expr` expressions consisting of && and == operators only, by deriving and adding a list of internal power

states to the PST. The multiple derived states will achieve the same OR logic specified. These internal derived states will not be written to in the output UPF and will be used only for internal purposes.

Example 1

In the following example, the `-supply_expr` contains one || operator of two == operations:

```
add_power_state sst -state SST_OFF
 {-supply_expr { power == {OFF} || ground == {OFF} } }
```

The tool internally converts the preceding command to the following:

```
add_power_state sst -state SST_OFF
 {-supply_expr { power == {OFF} } }

add_power_state sst -state SST_OFF_snps_1
 {-supply_expr { ground == {OFF} } }
```

Two hidden power states `SST_OFF` and `SST_OFF_snps_1` will be created, one for each == operation:

- Hidden state `SST_OFF`: `power == OFF`
- Hidden state `SST_OFF_snps_1`: `ground == OFF`

Example 2

In this example, the `-supply_expr` contains two || operators and one && operator:

```
add_power_state sst -state SST_HI
 {-supply_expr { (power == {FULL_ON 1.2} || power == {FULL_ON 0.7}) &&
                (ground == {FULL_ON 0.0} || nwell == {FULL_ON 0.5}) } }
```

The tool internally converts the preceding command to the following:

```
add_power_state sst -state SST_HI
 {-supply_expr { power == {FULL_ON 1.2} && ground == {FULL_ON 0.0} } }
add_power_state sst -state SST_HI_snps_1
 {-supply_expr { power == {FULL_ON 0.7} && ground == {FULL_ON 0.0} } }
add_power_state sst -state SST_HI_snps_2
 {-supply_expr { power == {FULL_ON 1.2} && nwell == {FULL_ON 0.5} } }
add_power_state sst -state SST_HI_snps_3
 {-supply_expr { power == {FULL_ON 0.7} && nwell == {FULL_ON 0.5} } }
```

After conversion, the input is equivalent to following four hidden power states OR-ed together:

- Hidden state `SST_HI`: `power == 1.2 && ground == 0.0`
- Hidden state `SST_HI_snps_1`: `power == 0.7 && ground == 0.0`

Feedback

- Hidden state `SST_HI_snps_2`: `power == 1.2 && nwell == 0.5`
- Hidden state `SST_HI_snps_3`: `power == 0.7 && nwell == 0.5`

Example 3

When a power state table using `add_power_state -group` or `-domain` involves a supply state from which internal states are created, these internal states will be used to create the internally derived PST. See the following example:

```
add_power_state SS1 -state NORMAL {-supply_expr
 {(power == {FULL_ON 1.2} || power == {FULL_ON 0.7}) &&
 ground == {FULL_ON 0.0}}}

add_power_state SS2 -state TURBO {-supply_expr
 {(power == {FULL_ON 1.0} || power == {FULL_ON 0.8}) &&
 ground == {FULL_ON 0.0}}}

create_power_state_group PSG

add_power_state -group PSG -state FAST
 {-logic_expr  { SS1 == NORMAL && SS2 == TURBO } }
```

The following table shows the internal representation of the derived PST.

Internal derived PST	SS1	SS2
FAST [NORMAL] [TURBO]	1.2	1.0
FAST [NORMAL_snps_1] [TURBO]	0.7	1.0
FAST [NORMAL] [TURBO_snps_1]	1.2	0.8
FAST [NORMAL_snps_1] [TURBO_snps_1]	0.7	0.8

Operator Precedence

The supply expression will follow the operator precedence as described in the IEEE 1801-3.1 standard. See the following tables. The precedence between the same class of operators will follow operator precedence mentioned in the SystemVerilog LRM.

Table 6 Boolean Operators

Operator	SystemVerilog equivalent	VHDL equivalent	Meaning
!	!	`not`	Logical negation

Feedback

Table 6 Boolean Operators (Continued)

Operator	SystemVerilog equivalent	VHDL equivalent	Meaning
~	`~`	`not`	Bit-wise negation
==	`==`	`=`	Equal
!=	`!=`	`/=`	Not equal
&&	`&&`	`and`	Logical conjunction
||	`||`	`or`	Logical disjunction

Table 7 Operator Precedence

Operator	Precedence
! ~	Highest
== !=	Next highest
&&	Next highest
||	Lowest

Reporting Support

`report_pst`, `report_supply_sets`, and `save_upf` will report original power state names and voltage values corresponding to those states.

See the following `report_pst -derived` example:

```
# User input
add_power_state SS1 -state NORMAL
 {-supply_expr {(power == {FULL_ON 1.2} || power == {FULL_ON 1.1})
 && ground == {FULL_ON 0.0}}}

add_power_state SS2 -state TURBO \
 {-supply_expr {(power == {FULL_ON 1.0} || power == {FULL_ON 0.8})
 && ground == {FULL_ON 0.0}}}

create_power_state_group PSG

add_power_state -group PSG -state FAST   {-logic_expr
 { SS1 == NORMAL && SS2 == TURBO } }

# Output of report_pst -derived
------------------------------------------------------------
```

```
PST Name    :  <Derived PST>

------------------------------------------------------------

Supplies    :  SS1[p][g]              | SS2[p][g]

States

 (1) <drv> :  NORMAL [1.20][0.00]   | TURBO [1.00][0.00]

 (2) <drv> :  NORMAL [1.10][0.00]   | TURBO [1.00][0.00]

 (3) <drv> :  NORMAL [1.20][0.00]   | TURBO [0.80][0.00]

 (4) <drv> :  NORMAL [1.10][0.00]   | TURBO [0.80][0.00]

------------------------------------------------------------
```

Specifying Logic Expressions

The `-logic_expr` option of the `add_power_state` command references control signals, clock signal intervals, and power states of an object. The option is parsed but ignored if it is used without the `-group` option. If a state has only logic signals defined in the logic expression, the state is parsed, ignored, and written to the output UPF. The logic expression uses the following syntax:

```
object == state && object == state
```

Only binary values (0 or 1) are allowed as logic signals. In addition, the use of parentheses is supported. For example:

```
add_power_state -domain PD1
  -state {ON -logic_expr {(SS1 == ON) && (nPWRUP == 1)}}
```

If the `-logic_expr` option is used with supply sets, you can use all operators listed in the IEEE 1801 language reference in the Boolean expression, as shown in Table 8.

Table 8 Boolean Operators

Operator	Meaning
!	Logical negation
~	Bitwise negation
<	Less than
<=	Less than or equal

Feedback

Table 8 Boolean Operators (Continued)

Operator	Meaning
>	Greater than
>=	Greater than or equal
==	Equal
!=	Not equal
&	Bitwise conjunction
^	Bitwise exclusive disjunction
|	Bitwise disjunction
&&	Logical conjunction
||	Logical disjunction

You can also use the || (OR) operator for operands of type logic control signals in the `-logic_expr` of `add_power_state` for supply set objects. See the following example:

```
add_power_state ss_peri_sw
  -state ON  {-logic_expr {RET_A == 1'b1 || PDW_A == 1’b1} }
  -state OFF {-logic_expr {RET_B == 1'b0 || PDW_B == 1’b0} }
```

Note:

The || operator involving supply set states is not supported in `-logic_expr` of `add_power_state`. Only || operators involving logical pins/ports are supported in `-logic_expr`.

Logic Expressions With add_power_state -group

If `add_power_state -group` involves a power state from which hidden power states are created (due to the use of the || operator in `-supply_expr`, see Specifying Supply Expressions), all of these hidden power states will be used to create the internally derived PST. See the following example:

```
# User input
add_power_state ss1 -state SS1_HI {-supply_expr
 { (power == {FULL_ON 1.2} || power == {FULL_ON 0.7}) &&
    ground == {FULL_ON 0.0} } }
add_power_state ss2 -state SS2_HI {-supply_expr
```

```
 { (power == {FULL_ON 1.2} || power == {FULL_ON 0.7}) &&
    ground == {FULL_ON 0.0} } }
create_power_state_group psg
add_power_state -group psg -state FAST   {-logic_expr
 { ss1 == SS1_HI && ss2 == SS2_HI } }

# Internal conversion by the tool
add_power_state ss1 -state SS1_HI        {-supply_expr
 { power == {FULL_ON 1.2} && ground == {FULL_ON 0.0} } }
add_power_state ss1 -state SS1_HI_snps_1 {-supply_expr
 { power == {FULL_ON 0.7} && ground == {FULL_ON 0.0} } }
add_power_state ss2 -state SS2_HI        {-supply_expr
 { power == {FULL_ON 1.2} && ground == {FULL_ON 0.0} } }
add_power_state ss2 -state SS2_HI_snps_1 {-supply_expr
 { power == {FULL_ON 0.7} && ground == {FULL_ON 0.0} } }
add_power_state -group psg -state FAST   {-logic_expr
 { ss1 == SS1_HI && ss2 == SS2_HI } }
```

The power state group `psg` will remain as is with no internal conversion. When the tool processes `SS1_HI` for power state group `psg`, both the `SS1_HI` and `SS1_HI_snaps_1` hidden power states will be added to the derived PST. The same is true for `SS2_HI`.

Supply and Logic Expressions With add_power_state -update

When a power state for a supply is defined with a `-supply_expr` and `-logic_expr`, and subsequently updated using `-update`, its definition becomes the conjunction of two:

```
supply_expr' = (previous supply_expr) && (-update supply_expr)

logic_expr’ = (previous logic_expr) && (-update logic_expr)
```

See the following example:

```
# User input

add_power_state SS1 -state ON {-supply_expr { power == {FULL_ON 1.2}
  || power == {FULL_ON 1.1}}}

add_power_state SS1 -state ON {-supply_expr { ground == {FULL_ON 0.0}
  || nwell == {FULL_ON 0.7}}} -update

Internally derived states

add_power_state SS1 -state ON
   {-supply_expr { power == {FULL_ON 1.2} && ground == {FULL_ON 0.0} }}

add_power_state SS1 -state ON_snps_1
   {-supply_expr { power == {FULL_ON 1.1} && ground == {FULL_ON 0.0} }}

add_power_state SS1 -state ON_snps_2
   {-supply_expr { power == {FULL_ON 1.2} && nwell == {FULL_ON 0.7} }}
```

Feedback

```
add_power_state SS1 -state ON_snps_3
   {-supply_expr { power == {FULL_ON 1.1} && nwell == {FULL_ON 0.7} }}
```

Successive Refinement of Power States

Using the `-update` option of the `add_power_state` command, you can incrementally add more information to an already defined state. The tool supports successive refinement for all types of objects that are supported by the `add_power_state` command.

The general rules for refinement using the `-update` option are as follows:

- The initial definition of the state can contain just the name of the state. It might or might not specify any additional information.
- If a state defined with the `-supply_expr` option is updated with another `-supply_expr` option, the new definition is the conjunction of the two definitions.

  You can update the supply expression in the following two ways:

  - If the initial definition contains just the netstate (`FULL_ON` or `OFF`) for a functional net without voltage, you can update the definition with the voltage.

    ```
    add_power_state -supply SS1 -state ON \
        {-supply_expr {power == FULL_ON}}}
    add_power_state -supply SS1 -state ON \
        {-supply_expr {power == FULL_ON 1.0}}} -update
    ```

  - The initial definition might be one or more functional nets. You can update the definition by specifying more functional nets.

    ```
    add_power_state -supply SS1 -state ON \
        {-supply_expr {power == FULL_ON 1.0}
    add_power_state -supply SS1 -state ON \
        {-supply_expr {ground == FULL_ON 0.0}}} -update
    ```

- If a state defined with the `-logic_expr` option is updated with another `-logic_expr` option, then the new definition is interpreted as the conjunction of the two expression. For example, the following statements

  ```
  add_power_state SS1 -state ON {-logic_expr {net1}}
  add_power_state SS1 -state ON (-logic_expr {net2}} -update
  ```

  are interpreted as

  ```
  add_power_state SS1 -state ON {-logic_expr {net1 && net2}}
  ```

- If the simstate is specified as `NOT_NORMAL`, you can update it to any simstate other than `NORMAL`.
- If the simstate is specified as `NORMAL`, `CORRUPT`, or `CORRUPT_ON_ACTIVITY`, you cannot update it to any other simstate.

Feedback

- A partially defined state can be used in the logic expression of a group or domain.
- You cannot change a legal state to an illegal state.

Feedback

Creating Power State Tables

To create a power state table, use the `add_power_state` and `create_power_state_group` commands.

Assuming that the supply sets are already defined, create power states for each supply set:

```
add_power_state \
    -supply SS1 -state ON  {-supply_expr {power == {FULL_ON 0.8} \
                                      && {ground == {FULL_ON 0}}}
add_power_state \
    -supply SS2 -state ON  {-supply_expr {power == {FULL_ON 0.8}}} \
                -state OFF {-supply_expr {{power == {OFF}}}
add_power_state \
    -supply SS3 -state ON  {-supply_expr {power == {FULL_ON 0.8}}}} \
                -state OFF {-supply_expr {{power == {OFF}}}
```

Next, create a power state group as follows:

```
create_power_state_group MY_PST
```

Finally, build the power state table using the specified states and the power state group:

```
add_power_state -group MY_PST \
    -state RUN12 {-logic_expr {SS1 == ON && SS2 == ON && SS3 == ON}} \
    -state RUN1  {-logic_expr {SS1 == ON && SS2 == ON && SS3 == OFF}} \
    -state RUN2  {-logic_expr {SS1 == ON && SS2 == OFF && SS3 == ON}} \
    -state SLEEP {-logic_expr {SS1 == ON && SS2 == OFF && SS3 == OFF}}
```

The power state table you built is shown in Table 9.

Table 9 Power State Table for MY_PST

State	SS1	SS2	SS3
RUN12	ON	ON	ON
RUN1	ON	ON	OFF
RUN2	ON	OFF	ON
SLEEP	ON	OFF	OFF

Using internal state names, the tool builds the actual power state table as shown in Table 10.

Feedback

Table 10 Power State Table Using Internal State Names

State	SS1.ground	SS1.power	SS2.power	SS3.power
RUN12	SNPS_INT_ON_2	SNPS_INT_ON_1	SNPS_INT_ON_3	SNPS_INT_ON_5
RUN1	SNPS_INT_ON_2	SNPS_INT_ON_1	SNPS_INT_ON_3	SNPS_INT_ON_6
RUN2	SNPS_INT_ON_2	SNPS_INT_ON_1	SNPS_INT_ON_4	SNPS_INT_ON_5
SLEEP	SNPS_INT_ON_2	SNPS_INT_ON_1	SNPS_INT_ON_4	SNPS_INT_ON_6

Hierarchical Power State Tables

When creating power state tables, you can build them hierarchically using existing power state tables and combining them to form a larger table. Suppose you want to increase the number of supply sets in Table 9 to add supplies and states as shown in Table 11.

Table 11 Adding Supplies to the Power State Table

State	SS1.ground	SS1.power	SS2.power	SS3.power	SS4.power	SS5.power
RUN12_NORM	GND	ON	ON	ON	ON	ON
RUN12_OVD	GND	ON	ON	ON	OVD	ON
RUN1_NORM	GND	ON	ON	OFF	ON	ON
RUN1_UND	GND	ON	ON	OFF	ON	OFF
RUN2_NORM	GND	ON	OFF	ON	ON	ON
RUN2_UND	GND	ON	OFF	ON	ON	OFF
SLEEP_OFF	GND	ON	OFF	OFF	OFF	OFF

The new power state table can be built hierarchically using the existing MY_PST table and creating a new power state table with new states.

First, create a new power group:

```
create_power_state_group MY_PST2
```

Feedback

Next, create the power states for the supplies in the new power group:

```
add_power_state -group MY_PST2 \
    -state NORM {-logic_expr {SS4==ON && SS5==ON}} \
    -state OVD  {-logic_expr {SS4==OVD && SS5==ON}} \
    -state UND  {-logic_expr {SS4==ON && SS5==OFF}} \
    -state OFF  {-logic_expr {SS4==OFF && SS5==OFF}}
```

These power states combine to form a new power state table as shown in Table 12.

Table 12 Power State Table MY_PST2

State	SS4	SS5
NORM	ON	ON
OVD	OVD	ON
UND	ON	OFF
OFF	OFF	OFF

Finally, create the new combined power state table:

```
create_power_state_group SYSTEM_PST
add_power_state -group SYSTEM_PST
    -state RUN12_NORM {-logic_expr {MY_PST==RUN12 && MY_PST2==NORM}} \
    -state RUN12_OVD  {-logic_expr {MY_PST==RUN12 && MY_PST2==OVD}} \
    -state RUN1_NORM  {-logic_expr {MY_PST==RUN1 && MY_PST2==NORM}} \
    -state RUN1_UND   {-logic_expr {MY_PST==RUN1 && MY_PST2==UND}} \
    -state RUN2_NORM  {-logic_expr {MY_PST==RUN2 && MY_PST2==NORM}} \
    -state RUN2_UND   {-logic_expr {MY_PST==RUN2 && MY_PST2==UND}} \
    -state SLEEP_OFF  {-logic_expr {MY_PST==SLEEP && MY_PST2==OFF}}
```

The power state table, SYSTEM_PST, is shown in Table 13. This table is equivalent to the power state table in Table 11.

Table 13 Power State Table SYSTEM_PST

State	MY_PST	MY_PST2
RUN12_NORM	RUN12	NORM
RUN12_OVD	RUN12	OVD
RUN1_NORM	RUN1	NORM

Feedback

Table 13 Power State Table SYSTEM_PST (Continued)

State	MY_PST	MY_PST2
RUN1_UND	RUN1	UND
RUN2_NORM	RUN2	NORM
RUN2_UND	RUN2	UND
SLEEP_OFF	SLEEP	OFF

Support for add_power_state with OR Operators in Top-Down Hierarchical Flow

In a top-down hierarchical flow, a portion of system PST gets partitioned into the block as derived PST and derived group. Supply states with the || operator in `-supply_expr`, which are used in power state groups, will be included in BLOCK UPF as newly derived states.

`split_constraints` adds support for `add_power_state` with OR operators. See the following example.

Consider the following input UPF:

```
add_power_state SST -state ON1
        {-supply_expr {power == `{FULL_ON, 1.0} &&
        ground == `{FULL_ON, 0.0}}
add_power_state mid1/SS1 -state ON1
        {-supply_expr {power == `{FULL_ON, 0.9} &&
        ground == `{FULL_ON, 0.0}}}
add_power_state mid2/SS2 -state ON1
        {-supply_expr {power == `{FULL_ON, 0.9} ||
        power == `{FULL_ON, 1.0}}}
# mid1/SS1 and mid2/SS2 are associated
add_power_state -group GRP -state S1
        {-logic_expr {SST == ON1 && mid1/SS1 == ON1 && mid2/SS2 == ON1}}
```

From `split_constraints`, we have the following:

Example 10 mid1.upf

```
add_power_state SST -state ON1
     {-supply_expr {power == `{FULL_ON, 1.0} &&
     ground == `{FULL_ON, 0.0}}
add_power_state SS1 -state SNPS_mid1_ON1
     {-supply_expr {power == `{FULL_ON, 0.9} &&
     ground == `{FULL_ON, 0.0}}}
add_power_state SS1 -state SNPS_mid2_ON1
     {-supply_expr {power == `{FULL_ON, 0.9} ||
     power == `{FULL_ON, 1.0}}}
```

Feedback

```
add_power_state -group GRP -state S1
     {-logic_expr {SST == ON1 && SS1 == SNPS_mid1_ON1 &&
     SS1 == SNPS_mid2_ON1}}
```

Example 11 mid2.upf

```
add_power_state SST -state ON1
     {-supply_expr {power == `{FULL_ON, 1.0} &&
     ground == `{FULL_ON, 0.0}}
add_power_state SS2 -state SNPS_mid1_ON1
     {-supply_expr {power == `{FULL_ON, 0.9} &&
     ground == `{FULL_ON, 0.0}}}
add_power_state SS2 -state SNPS_mid2_ON1
     {-supply_expr {power == `{FULL_ON, 0.9} ||
     power == `{FULL_ON, 1.0}}}
add_power_state -group GRP -state S1
     {-logic_expr {SST == ON1 && SS2 == SNPS_mid1_ON1 &&
     SS2 == SNPS_mid2_ON1}}
```

Note:

- Supply sets mid1/SS1 and mid2/SS2 are associated in the top.
- The effective power state on this supply net group has this supply expression:
  - `{power == `{FULL_ON, 0.9} && ground == `{FULL_ON, 0.0}}`
- If power states on mid1/SS1 and mid2/SS2 have different names, block UPF will use their original power state names.
- If power states on mid1/SS1 and mid2/SS2 have the same names, block UPF will use the tool-generated power state names in the format `SNPS_`*`block name_original power state name`* to distinguish them.
- In mid1.upf, supply set SS1 has two power states with tool-genenrated names `SNPS_mid1_ON1` (which comes from power state ON1 on mid1/SS1) and `SNPS_mid2_ON1` (which comes from power state ON1 on mid2/SS2).
- In mid1.upf, power state group GRP has a logic expression with two `SS1 == xxx` operands in the && expression. The intersection is exactly `{power == `{FULL_ON, 0.9} && ground == `{FULL_ON, 0.0}}`.
- Hidden power state names should not appear in the block UPF.
- `-supply_expr` with OR operators should appear in the block UPF in exactly in the same way as in the original UPF.

Feedback

Creating Power State Groups in Hierarchies Having State Propagation Enabled

You can create power state groups in hierarchies having state propagation enabled. That is, when you have a BLOCK with state propagation disabled and a TOP design with state propagation enabled, you can create power state groups in TOP and use these groups to refer to the states in BLOCK, to define the complete power state table (PST).

From the groups created in TOP, you can refer to all the following four types of states defined in BLOCK:

- PST states
- Supply set states
- Domain states
- Other group states

Groups, however, cannot refer to port states added using the `add_port_state` command and net states added using the `add_supply_state` command. You must use `create_pst` to refer to port states or net states.

Defining domain states in TOP is identical to defining group states. You can create domain states too in TOP UPF similar to group states.

Example

The following example shows how to create a power state group in TOP and define the relationship between group state of BLOCK and TOP PST states.

Block has group state GS in group GROUP_B.

Top has PST TOP_PST having power state PS1.

The group GROUP_T in top is referring to block's group state GS and top's PST state PS1 in power state ST1.

```
# BLOCK.upf
set_design_attributes -elements {.} -attribute
 enable_state_propagation_in_add_power_state FALSE
...
add_power_state SS -state ON {-supply_expr {power=={FULL_ON 1} &&
 ground=={FULL_ON 0}}}
...
create_power_state_group GROUP_B
add_power_state GROUP_B -state GS {-logic_expr {SS==ON && SS1==ON}}

# TOP.upf
set_design_attributes -elements {.} -attribute
 enable_state_propagation_in_add_power_state TRUE
```

Feedback

```
...
create_pst TOP_PST -supplies           {VDD VDD1}
add_pst_state PS1 -pst TOP_PST -state {ON  ON}

create_power_state_group GROUP_T
add_power_state GROUP_T -state ST1 {-logic_expr {TOP_PST==PS1 &&
 BLOCK/GROUP_B==GS}}
```

Bottom-Up and Top-Down Hierarchical Flows

Bottom-Up Flow

In this tool, you will use the `link` command to propagate UPF constraints from a synthesized BLOCK to TOP, as part of the bottom-up hierarchical flow. In this flow, with state propagation enabled for BLOCK, group states in BLOCK will be propagated to TOP with/without domain merging differently.

Without Domain Merging

In this flow, the tool propagates groups created in BLOCK (with state propagation enabled) to TOP. The state propagation value of TOP is irrelevant.

With Domain Merging

Consider a scenario where BLOCK has state propagation enabled. (It is irrelevant whether TOP has state propagation enabled or disabled.) Consider you have created domain states in BLOCK. During domain merging, domain states in BLOCK and the supply sets are deleted when they are merged with the TOP domain. In this scenario, to preserve the domain states defined in BLOCK, to ensure that the system PST is not impacted, the tool:

- Creates a new group by name “<domain_name>” in BLOCK
- Transfers the domain states in BLOCK to the newly created group

Top-Down Flow

Currently, the tool does not support this functionality in `split_constraints` in top-down hierarchical flow synthesis.

Hierarchical Flow Example

Consider this hierarchical design example with TOP > BLOCK [MID > BOT].

In this example, the hierarchical design TOP has a lower hierarchical BLOCK. TOP has the PDTOP power domain with the `merge_domain` design attribute set to true. The BLOCK has the PDTOP power domain for BLOCK scope with domain states and another lower scope PST called MID_PST.

```
# MID.upf
set_design_attributes -elements {.}
 -attribute enable_state_propagation_in_add_power_state TRUE
```

Feedback

```
...
create_pst MID_PST
add_pst_state s0 -pst MID_PST
set_scope BOT
set_design_attributes -elements {.}
 -attribute enable_state_propagation_in_add_power_state FALSE
...
add_power_state SS -state ON
 {-supply_expr {power=={FULL_ON 1} && ground=={FULL_ON 0}}}
...
set_scope ../
create_power_domain PDTOP
add_power_state PDTOP -state PS1
   {-logic_expr {MID_PST==s0 && BOT/SS==ON}}

# TOP.upf
set_design_attributes -elements {.}
 -attribute enable_state_propagation_in_add_power_state TRUE
create_power_domain PDTOP
set_design_attributes -elements {MID} -attribute merge_domain TRUE
```

During `propagate_constraints`, the PDTOP domain of BLOCK is merged with the TOP level PDTOP domain and the group states of BLOCK are propagated to TOP.

The domain states of BLOCK's domain are retained in BLOCK by creating another power state group called PDTOP.

```
# TOP.upf  [full chip UPF]
create_power_domain PDTOP
set_scope MID
set_design_attributes -elements {.}
 -attribute enable_state_propagation_in_add_power_state TRUE
...
create_pst MID_PST
add_pst_state s0 -pst MID_PST
set_scope BOT
set_design_attributes -elements {.}
 -attribute enable_state_propagation_in_add_power_state FALSE
...
add_power_state SS -state ON
 {-supply_expr {power=={FULL_ON 1} && ground=={FULL_ON 0}}}
...

# Back to MID scope, new group created in block scope
set_scope ../
create_power_state_group PDTOP
add_power_state PDTOP -state PS1
   {-logic_expr {MID_PST==s0 && BOT/SS==ON}}
```

Feedback

Reconciling Voltages When Building System Power State Tables

In a hierarchical design, the tool creates a final system power state table after checking the consistency between power states and power state tables in the top-level and block-level power state tables. By default, any inconsistency in voltages and supply states for power state tables results in the tool dropping that state.

However, you can continue with the tool flow without dropping states by allowing the tool to perform voltage reconciliation, either globally or for specific block boundaries.

To enable voltage reconciliation across all hierarchical boundaries with an infinite voltage matching tolerance, use the following command:

```
fc_shell> set_design_attributes -elements {.} \
          -attribute upf_reconciliation_automatic true
```

This specification takes precedence over any block-specific attributes and thresholds. The tool considers every hierarchical boundary to be a reconciliation boundary. In a top-down flow, the power states of the higher scope power state table are pushed to the lower scope power state tables. In a bottom-up flow, the tool checks consistency between attributes pushed from the block level to the top level.

To specify voltage reconciliation for specific block boundaries, set the `upf_reconcile_boundary` design attribute to mark the block boundaries. You can then specify that the tool skip all power state table checking below the boundary hierarchy, as shown in the following command:

```
fc_shell> set_design_attributes -models IP1 \
          -attribute upf_reconcile_boundary "skip"
```

Alternatively, you can specify that the tool reconcile voltages for the power state tables. For example, the following command marks a block boundary for voltage reconciliation on all power state tables below the boundary hierarchy:

```
fc_shell> set_design_attributes -models IP1 \
          -attribute upf_reconcile_boundary "reconcile_voltages"
```

If you reconcile the voltages, you can set a voltage tolerance threshold that defines a range of acceptable voltages for block-level states. For example, the following command specifies a single threshold tolerance on a supply:

```
fc_shell> set_variation -supply {BLK/VDD} -tolerance {0.2}
```

The following command sets positive and negative thresholds on a supply:

```
fc_shell> set_variation -supply {BLK/VDD} -tolerance {0.2 0.4}
```

The following command specifies a global threshold on the entire design:

```
fc_shell> set_variation -tolerance {0.2}
```

Feedback

Example of Reconciling Voltages

Suppose a top-level power state table, as shown in Table 14, and a block-level power state table, as shown in Table 15, have similar supply sets.

Table 14 Top-Level Power State Table

State	VDD1	VDD2	VDD3	VDD4
S0	1.00	1.4	0.75	0.90
S1	1.25	1.4	0.85	1.15
S2	0.85	1.3	0.85	1.15
S3	OFF	1.3	0.85	1.15

Table 15 Block-Level Power State Table

State	VDD1	VDD2	VDD3	VDD4
S0	1.00	1.4	0.75	0.90
S1	1.25	OFF	0.85	1.15
S2	0.90	1.2	0.75	1.25

The following voltage threshold variations is set for the block-level states:

```
set_variation -tolerance {0.1}
```

The block-level power state table is expanded in Table 16.

Table 16 Block-Level Power State Table After Range Expansion

State	VDD1	VDD2	VDD3	VDD4
S0	0.9 - 1.1	1.3 - 1.5	0.65 - 0.85	0.8 - 10
S1	1.15 - 1.35	OFF	0.75 - 0.95	1.05 - 1.25
S2	0.8 - 1.0	1.1 - 1.3	0.65 - 0.85	1.15 - 1.35

The top-level S0 state is aligned with the block-level S0 state. The top-level S1 state is dropped due to a conflict in the block-level S1 state's VDD2. Top-level S2 is aligned with

Feedback

block-level S2; and top-level S3 is dropped due to a conflict in VDD1. The final system power state table is shown in Table 17.

Table 17 Final System Power State Table After Voltage Reconciliation

State	VDD1	VDD2	Vdd3	VDD4
S0	1.00	1.4	0.75	0.90
S2	0.85	1.3	0.85	1.15

Reporting Voltage Reconciliation Constraints

To display the voltage thresholds set on your supplies, use the `report_pst -reconcile` command. The report contains a section with all the specified voltage thresholds, as shown in the following example. In most cases, the reported state names are user-defined state names. However, if you use the `-supplies` option with the `report_pst` command, the report displays tool-derived power state names for the specified supply nets.

```
fc_shell> report_pst -verbose -reconcile
------------------------------------------------------
Reconciliation skipped blocks
           :  N/A
------------------------------------------------------
Reconciliation on blocks
           :  mid
------------------------------------------------------
Reconciliation thresholds
Global     :  (-, -)
Threshold applied on supplies :
SS1.power  : (-0.10, +0.10)
SS1.ground : (-,-)
SS2.power  : (-0.10, +0.10)
SS2.ground : (-,-)
------------------------------------------------------

Scope      :  top
------------------------------------------------------
Supply names : SS1, SS1, SS2, SS2
Resulting power states in this scope:

          SS1|   SS1|   SS2|  SS2|
          [p]|   [g]|   [p]|  [g]|
 drv :    ON1|   ON4|   ON1|  ON5|
 drv :    ON2|   ON4|   ON2|  ON5|
------------------------------------------------------
```

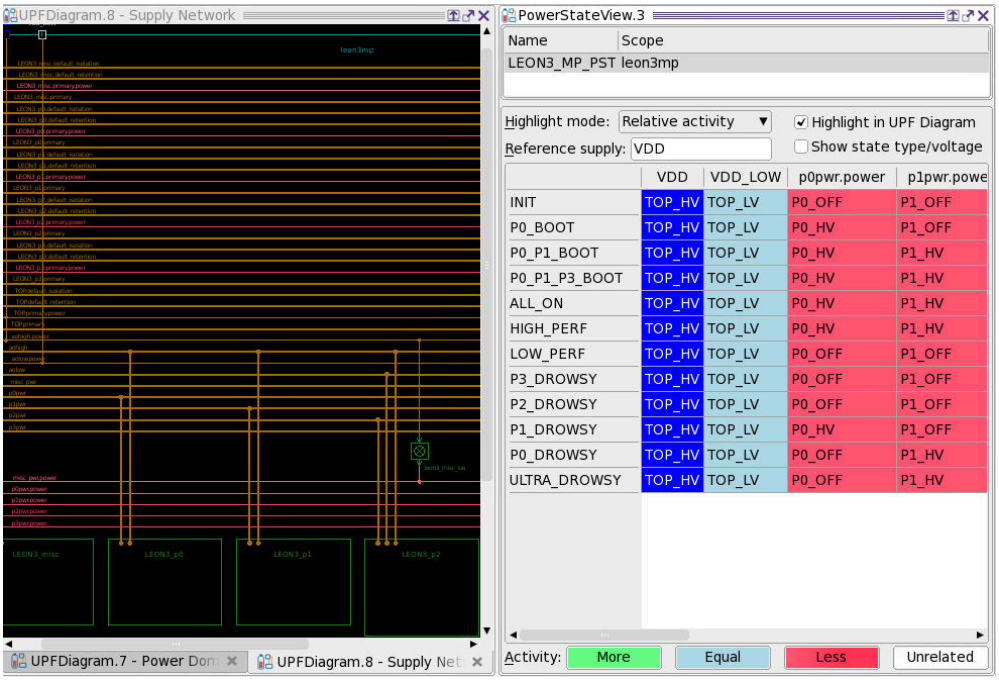
Feedback

Visually Analyzing Power State Tables in the GUI

To analyze and debug the isolation and level-shifter strategies in a UPF design, use the UPF diagram view with the Power State Table panel. You can view the power states for each supply in a power state table and examine their relationships in the Supply Network View.

Figure 46 shows an example of the Supply Network View and Power State Table View during always-on analysis.

Figure 46 Always-On Analysis Using the Power State Table View



You can display the power state table by choosing View > Multivoltage Views > Power State.

The Power State Table view provides the following types of analysis:

- Always-on analysis compares the on-off states between a supply (highlighted in dark blue) and the rest of the supplies in the power state table. In this figure, all the supplies are compared to VDD. VDD_LOW is equally on while p0pwr.power is less on than VDD.
- The supplies are highlighted in the supply network view with the corresponding colors.

Feedback

Power Models

The Fusion Compiler tool supports the IEEE 1801 construct called a power model. A power model allows you to define the UPF for a hard macro in a self-contained environment. Each time this macro is instantiated, the power model is loaded from memory instead of calling the `load_upf` command and `connect_supply_net` commands.

You define power models using the `define_power_model` command. The `apply_power_model` and `add_parameter` commands are also used when defining or specifying the use of power models.

To check your power models, use the `report_power_model` command. This command displays information including where the power models are defined and applied.

Topics covered in this section:

- Configuring Fusion Compiler for Power Models
- Defining and Applying a Power Model
- Excluding Designs From Using Power Models

Configuring Fusion Compiler for Power Models

To simply the loading of power model definitions, configure the following application options:

- `mv.upf.power_model_library`

  Specifies a list of all UPF power model names

- `mv.upf.power_model_search_path`

  Specifies a list of search paths that the tool uses to find the UPF power model

Defining and Applying a Power Model

Power models are IEEE 1801 commands encapsulated within the `define_power_model` command. The following is an example of power model definition for a power model named my_model:

```
define_power_model my_model -for {lib_cell_A lib_cell_B} {
   add_parameter DOMAIN -default "PD_TOP" -description "top power domain"
   create_supply_net VDD
   create_supply_net VSS
   create_supply_set SS -function {power VDD} -function {ground VSS}
   create_power_domain PD_$DOMAIN -supply {primary SS} -include_scope
}
```

The `add_parameter` command specifies the names of parameters to be defined inside a power model. These parameters can be overridden from the `apply_power_model` command.

After you define a power model, you can map the power model to a list of cell instances with the `apply_power_model` command. For example,

```
apply_power_model my_model -elements (u1/macro u2/macro)...
```

Excluding Designs From Using Power Models

If you want to exclude designs from using power models, do the following:

```
set_design_attributes -models model_list -is_hard_macro false
```

or

```
set_design_attributes -models model_list \
   -attribute UPF_is_hard_macro false
```

Setting UPF Attributes on Ports and Hierarchical Cells

To specify additional requirements for the ports and hierarchical cells of a power domain, use the `set_port_attributes` and `set_design_attributes` commands.

Topics in this section:

Feedback

Setting UPF Attributes on Ports

To set attributes and their values on the specified ports, use the `set_port_attributes` command. Table 18 shows the list of attributes and their values that can be specified on the ports using the `set_port_attributes` command.

Table 18 UPF Port Attributes

Attribute name	Attribute value	Ports where the attribute can be specified	Use of the attribute
`model`	Name of the object		Specifies a module or library cell to whose ports the attribute is to be applied.
`iso_sink`	Name of the supply set, `DERIVED_DIFF_ONLY`, `DERIVED_DIVERSE`	Output	Identifies the actual off-chip load of a primary output port.
`iso_source`	Name of the supply set, `DERIVED_DIFF_ONLY`, `DERIVED_DIVERSE`	Input	Identifies the actual off-chip driver of a primary input port.
`related_supply_default_primary`	Boolean	Top level input and output ports	Indicates that, when the related supply net is not specified, the primary supply of TOP domain is assumed as the related supply. Used by the verification tools so that no assumption is made about the default power supply.
`snps_derived`	Boolean	Input and output supply ports	Indicates that the specified ports have been created by Synopsys tools. You can specify this attribute in the bottom-up flow as well.
`repeater_power_netrepeater_ground_net`	Name of the supply net	Input and output ports and pins. Cannot be specified on bidirectional ports	Specifies a repeater (buffer) should be inserted to drive the specified port. The inserted buffer is powered by the specified supply net.

Feedback

Table 18 UPF Port Attributes (Continued)

Attribute name	Attribute value	Ports where the attribute can be specified	Use of the attribute
`driver_supply`		Input, output, and bidirectional ports. If specified on a bidirectional port, receiver and driver supplies should be electrically equivalent.	Specifies the supply for the logic driving the port
`receiver_supply`		Input, output, and bidirectional ports. If specified on a bidirectional port, the receiver and driver supplies should be electrically equivalent.	Specifies the supply for the logic reading the port

Note:

When you specify the `set_port_attributes` command multiple times on the same object, the last setting overrides the previous settings.

The following example shows how to set the `iso_source` and `iso_sink` attributes on the input and output ports, respectively.

```
fc_shell> set_port_attributes -ports {in1 in2} -attribute iso_source SS1
fc_shell> set_port_attributes -ports {out1 out2} -attribute iso_sink SS2
```

You can specify attributes using either the UPF 2.0 or 2.1 syntax. For example, either version is acceptable:

```
set_port_attributes -ports {in1 in2} -attribute iso_source SS1
set_port_attributes -ports {in1 in2} -attribute {iso_source SS1}
```

If the tool encounters an unrecognized port attribute when reading the UPF, it preserves the attribute in the output UPF.

Setting UPF Attributes on Hierarchical Cells

To set attributes on a collection of cells, use the `set_design_attributes` command. Table 19 shows the list of attributes and their values that can be specified on hierarchical cells using the `set_design_attributes` command.

Feedback

Table 19 UPF Design Attributes

Attribute name	Attribute value	Location of the attribute	Use of the attribute
`correlated_supply_group`	Supply net names or wildcard (*) character	Top scope of the design	Indicates that the supply nets of the port state or power state triplets should be considered as correlated voltage range.
`derived_external` and `external_supply_map`	Name of the supply set	Hierarchical cell	Indicates that the supply sets are reference-only supply sets. These are used with ports with the `iso_source` and `iso_sink` attributes. These attributes establish a one-to-one order dependent mapping of the supply sets..
`derived_iso_strategy`	Name of the isolation strategy	Hierarchical cell	To ensure unique strategy name for the derived strategies in the power domain. Used in hierarchical flow to support location fanout.
`enable_bias`	`true | false`	Top-level design	When set to `true`, turns on the well bias feature.
`legacy_block`	`true | false`	Hierarchical cell	When set to `true`, indicates that the block is a legacy block. This is useful while combining two blocks; one defined using domain-dependent supply nets and the other defined using domain-independent supply nets and supply sets. The block defined using domain-dependent supply nets is marked as a legacy block.
`lower_domain_boundary`	`true | false`	Top scope of the design and any hierarchical cell	When set to `true`, the boundary of the power domain extends up to the boundary of the power domain below it

Feedback

Table 19 UPF Design Attributes (Continued)

Attribute name	Attribute value	Location of the attribute	Use of the attribute
`merge_domain`	`true | false`	Hierarchical cell that is not the root cell of the power domain	Indicates that the two blocks belonging to the same power domain can be merged
`reference_only`	Supply set	Current design	Specifies a reference-only supply that cannot be used to power cells
`relax_constant_corruption_for_macro_inputs_and_primary_outputs`	`true | false`	Top-level design	Specifies whether to skip isolation cell insertion on literal constants directly driving macro inputs or primary output ports if the clamp value of the strategy matches the logic value of the constant.
`suppress_iss`	Power domain	Current design	Indicates that supply set handles cannot be created in the power domain
`upf_chip_design`	`true | false`	Top-level design	Indicates that the design is the TOP block. When the isolation strategy definition contains `-location fanout`, this attribute causes the primary output ports to be considered as the loads
`upf_reconcile_boundary`	`skip` or `reconcile_voltages`	Top-level design	Specifies whether to skip voltage checking or reconcile voltages when creating the system power state table.

You can specify design attributes using either the UPF 2.0 or 2.1 syntax. For example,

```
set_design_attributes -elements {m1} -attribute terminal_boundary true
set_design_attributes -elements {m1} -attribute {terminal_boundary true}
```

If the tool encounters an unrecognized design attribute when reading the UPF, it preserves the unrecognized attribute in the output UPF.

Feedback

Ungrouping Hierarchies that Drive Control Signals

When the control signal of a power management cell is set on a hierarchical port, the Fusion Compiler tool allows ungrouping the hierarchy of the driver pin of the control signal. The `ungroup_cells` command can ungroup hierarchies of hierarchical ports that are specified as `-isolation_signal` in the `set_isolation` command. Additionally, the `compile_fusion` command steps that perform ungrouping can also handle this scenario.

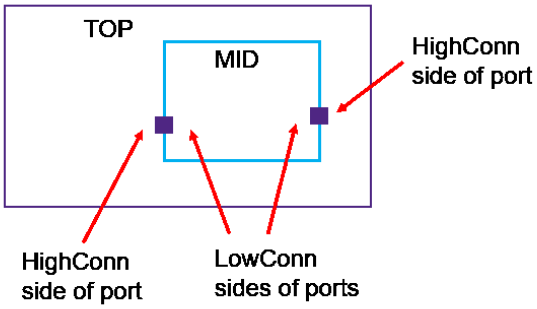
While ungrouping, the tool ensures that the connectivity of the enable signal with the power management cell is unaffected.

Setting UPF Attributes on Macros

Liberty definitions for hard macros might not contain sufficient information to model the interface between the macro and the top-level design.

In UPF terminology, a port has two sides: the HighConn side and the LowConn side, as shown in Figure 47. The HighConn side is visible to the parent of the instance whose interface contains the port. The LowConn side is visible inside the instance. In this example, TOP is the parent and MID is the instance.

Figure 47 Sides of a Port



To specify port properties, use the `set_related_supply_net` command and the `-driver_supply` and `-receiver_supply` options of the `set_port_attributes` command, as follows:

- If you apply the command to a port in the hierarchy at the scope of the command, the attribute applies to the LowConn side of the port. This attribute is used to implement the net connected to the LowConn side.
- If you apply the command to a port in the hierarchy below the scope of the command, the attribute applies to the HighConn side of the port. This attribute is used to implement the net connected to the HighConn side. If there is no attribute set on the HighConn side, the tool uses the attribute set on the LowConn side.

When the tool considers power intent logic insertion outside a hard macro, the receiver supply of an input port on the hard macro is determined in the following order of decreasing priority:

1. A `set_port_attributes -receiver_supply` command specified at the HighConn side of the port
2. A `set_related_supply_net` command specified at the HighConn side of the port
3. Any `related_power_port` information for the pin provided in the Liberty file

When the tool considers power intent logic insertion outside a black box cell, the receiver supply of an input port on the black box cell is determined in the following order of decreasing priority:

1. A `set_port_attributes -receiver_supply` command specified at the HighConn side of the port
2. A `set_related_supply_net` command specified at the HighConn side of the port
3. A `set_port_attributes -receiver_supply` command specified at the LowConn side of the port
4. A `set_related_supply_net` command specified at the LowConn side of the port
5. The primary supply of the power domain

The tool also checks any `set_port_attributes -driver_supply` commands specified at the HighConn side of the port for electrical violations.

If a discrepancy exists between the actual supply and the supply specified in `set_port_attributes` or `set_related_supply_net` commands, the tool issues one or more warning messages.

Feedback

Setting Design Attributes on Supply Nets and Logic Nets

The Fusion Compiler tool allows you to use the `set_design_attributes` command to apply user-defined attributes to supply nets and logic nets, which are the only UPF objects allowed in the argument of the `-elements` option.

This usage is primarily for the use of downstream tools. UPF files are used by multiple tools in a design flow, but each tool uses the UPF contents differently.

For example, the following command applies a user-defined attribute to a list of supply nets:

```
fc_shell> set_design_attributes -elements {VDD mid/VDD} \
   -attribute supply_net_attribute true
```

The following command applies a user-defined attribute to a list of logic nets:

```
fc_shell> set_design_attributes -elements {inst1/netA*} \
   -attribute logic_net_attribute true
```

Modeling Unconnected Pins on Macros

If the pins of a macro block do not have any related power or ground pins and the pins are not part of a feedthrough path, the tool implicitly assumes that the pins are unconnected. You can explicitly specify that a macro pin is unconnected in the following ways:

- Use the `set_port_attributes -unconnected -ports {...} -model {...}` command
- Set the Liberty `is_unconnected` attribute of the pin

Specifying Analog Nets

A net is an analog net if a pin on the net is an analog pin. You can specify analog pins in two ways:

1. Model the pin as an analog pin using the Liberty `is_analog` pin attribute
2. Use the `set_port_attributes -is_analog` command to specify an analog pin

Isolation, level-shifter, and repeater cells are not inserted on analog nets. No voltage checks for automatically inserted level-shifter cells are performed on analog nets. You should also set the `dont_touch` attribute on these nets to prevent buffering.

The `check_mv_design` command issues warning messages in the following cases:

- There is a voltage difference between connected analog pins
- A level-shifter, isolation strategy, or repeater strategy is specified on an analog net

- A digital pin is connected to an analog pin
- A buffer or inverter is present on an analog net
- An isolation cell or level shifter is present on an analog net
- Multiple instances of the same user-defined module (specified with the `-model` option) have different analog settings.

Specifying the Power Supply for a Literal Value

To specify the supply to use to model a literal value (1'b0 or 1'b1), use the `-literal_supply` option of the `set_port_attributes` command. This literal value is associated with a specified pin or port.

Legacy Blocks

A legacy block is a block designed before the introduction of supply sets, using only domain-dependent supply nets. A legacy block does not use or define any domain-independent supply nets or supply sets.

A conflict can arise when a legacy block is used in a design with domain-independent supply nets. To prevent such a conflict, set the block's `legacy_block` design attribute to `true`. This converts all power domains of the legacy block to be fully restricted, so the legacy block can no longer use any domain-independent supply nets declared in the scopes above the block.

A domain-independent supply net is a supply net that is available to any power domain defined at or below the scope of the supply net, as long such domains are not restricted. In other words, the supply net was created by a `create_supply_net` command without the `-domain` option. For example,

```
fc_shell> create_supply_net SN1
```

Conversely, a domain-dependent supply net is a supply net that is available only to the domain for which it is defined. In other words, the supply net was created by a `create_supply_net` command with the `-domain` option. For example,

```
fc_shell> create_supply_net SN2 -domain PD2
```

The supply net is created in the PD2 power domain and cannot be used in other domains.

A restricted power domain is a power domain that is restricted to use only certain supply sets. This restriction results from usage of the `extra_supplies` keyword with the `-supply` option of the `create_power_domain` command. For example,

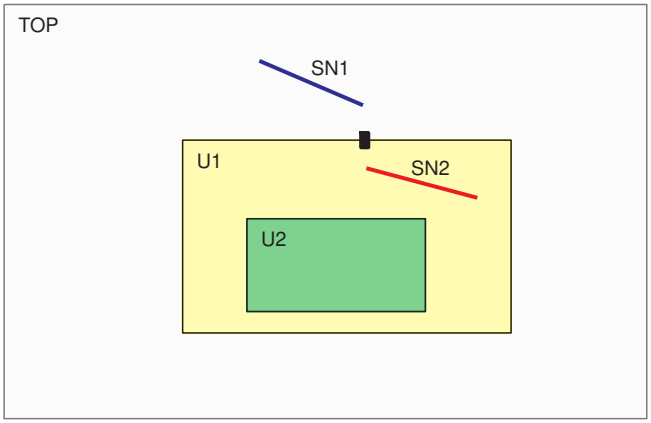
```
fc_shell> create_power_domain PD3 -elements U3 \
   -supply {extra_supplies_0 SS1}
```

Feedback

The PD3 power domain is restricted to using only the SS1 supply set.

When a legacy block is used in a newer design containing supply sets, a conflict can arise with a situation like the one shown in Figure 48.

Figure 48 Legacy Block Used in a Top-Level Design



In this diagram, U1 is a legacy block with a domain-dependent supply net, SN2. An instance of this block is used in the top-level design, TOP, which has a domain-independent supply net, SN1. U1 contains a lower-level block, U2. Because the SN1 supply net is domain-independent, it is available for use in all of the domains. On the other hand, because the SN2 supply net is domain-dependent, it is available for use only in the domain of block U1.

If the two supply nets are connected through a supply port on U1, the availability of the combined net in U2 is undefined. This might lead to the usage of the supply net in U2, which would be incorrect.

To clearly specify that this type of connection is not allowed, declare U2 to be a legacy block by using the following command:

```
fc_shell> set_design_attributes -elements U1 \
   -attribute legacy_block true
```

This command converts all power domains of the U1 block to fully restricted domains so that those domains can no longer use the domain-independent supply nets declared in scopes above the block. The tool achieves this effect by using the `-supply` option of the `create_power_domain` command when the block is read.

For example, the tool changes the following command:

```
create_power_domain PD -elements U2
```

to the following command:

```
create_power_domain PD -elements U2 -supply {extra_supplies ""}
```

Because there are no supply sets listed between the quotation marks in the `-supply` option, the domain becomes fully restricted and does not allow the usage of any supply sets defined at higher levels of the design, thereby preventing any supply set availability conflict from arising.

No domain-independent supply nets or supply sets can be defined or used inside a legacy block or any of its lower-level blocks, and no supply set handles can be used. Any lower-level blocks below a legacy block must also be legacy blocks.

Querying for UPF Design and Port Attributes

To query for UPF predefined attributes of the design objects, that is, attributes that are previously set on the design objects using `set_design_attributes` and `set_port_attributes`, use the `get_upf_design_attribute` and the `get_upf_port_attribute` commands.

The following table provides details on the attributes that can be queried using `get_upf_design_attribute`:

Attribute	Attribute Name	Object Type	Defined If	Value
-attribute {terminal_boundary}	UPF_terminal_boundary	instance	Always	TRUE, if the associated instance is marked as terminal boundary
-is_hard_macro	UPF_is_hard_macro	instance	Always	TRUE, if cell derived as a hard macro from Liberty or the set_design_attributes command. FALSE, otherwise
-is_soft_macro	UPF_is_soft_macro	instance	Always	TRUE, if cell is defined soft macro using set_design_attributes. FALSE, otherwise
enable_bias	UPF_enable_bias	instance	set_design_attributes -attribute enable_bias true is defined at top level	TRUE, if the given instance has enable_bias set to TRUE or if it sits within a block with enable_bias set to TRUE. FALSE, otherwise

The following table provides details on the attributes that can be queried on pin and port objects of the design using `get_upf_port_attributes`:

Attribute	Attribute name	Defined if	Return value
-driver_supply	UPF_driver_supply	Object is attributed driver supply **Note:** In the case of driver supply where multiple settings can be applied at the same time on an object based on scope, the returned value is the one with higher precedence.	Supply set
-receiver_supply	UPF_receiver_supply	Object is attributed with receiver supply **Note:** In the case of receiver supply where multiple settings can be applied at the same time on an object based on scope, the returned value is the one with higher precedence.	Supply set
-feedthrough	UPF_feedthrough	Object has a valid setting derived from set_port_attributes or Liberty	The collection of shorted pins
-unconnected	UPF_unconnected	Object is valid for set_port_attributes -unconnected setting	TRUE, if object is derived unconnected from set_port_attributes or Liberty. FALSE, otherwise
-clamp_value	UPF_clamp_value	Object has a valid clamp value setting	The clamp value attributed: 0 | 1 | latch
-is_analog	UPF_is_analog	Always	TRUE, if object is derived is_analog from set_port_attributes or Liberty. FALSE, otherwise

Feedback

Attribute	Attribute name	Defined if	Return value
-literal_supply	UPF_literal_supply	Object has a valid set_port_attributes -literal_supply_setting	A supply set, if the setting was supply set based. Otherwise, a pair of supply nets in the form {power ground}

Hierarchical Synthesis With ETMs and Macros

The following design objects provide only limited information to the tool:

- A macro is a reusable design block, which might be a user-created Liberty model or an IP provided by a vendor.
- An extracted timing model (ETM) is the Liberty model representation of a block, often generated using the `extract_model` command in the PrimePower or PrimeTime tools. An ETM captures the UPF information using relevant Liberty attributes.
- An empty RTL module is a design object created during design planning to represent an incomplete block. Such modules might contain different amounts and types of design information, such as a partial netlist or no netlist at all. These modules allow other operations in the design flow to continue until a more complete block design is available.

The Fusion Compiler tool treats macros, ETMs, and empty modules the same way. You can provide UPF information as follows:

- Use the `set_port_attributes` and `set_related_supply_net` commands from a design level outside the object.
- Specify UPF constraints outside the object in a scopeless flow.
- Specify a complete block UPF on the object scope.

Some of these design objects might not have PG pins or related pin information. In this case, the tool determines the driver and receiver supplies of the cell input ports as follows, in order of decreasing priority:

1. A `set_port_attributes -receiver_supply` command specified at the top-level scope
2. A `set_related_supply_net` command specified at the top-level scope
3. A `set_port_attributes -receiver_supply` command specified at the block-level scope

Feedback

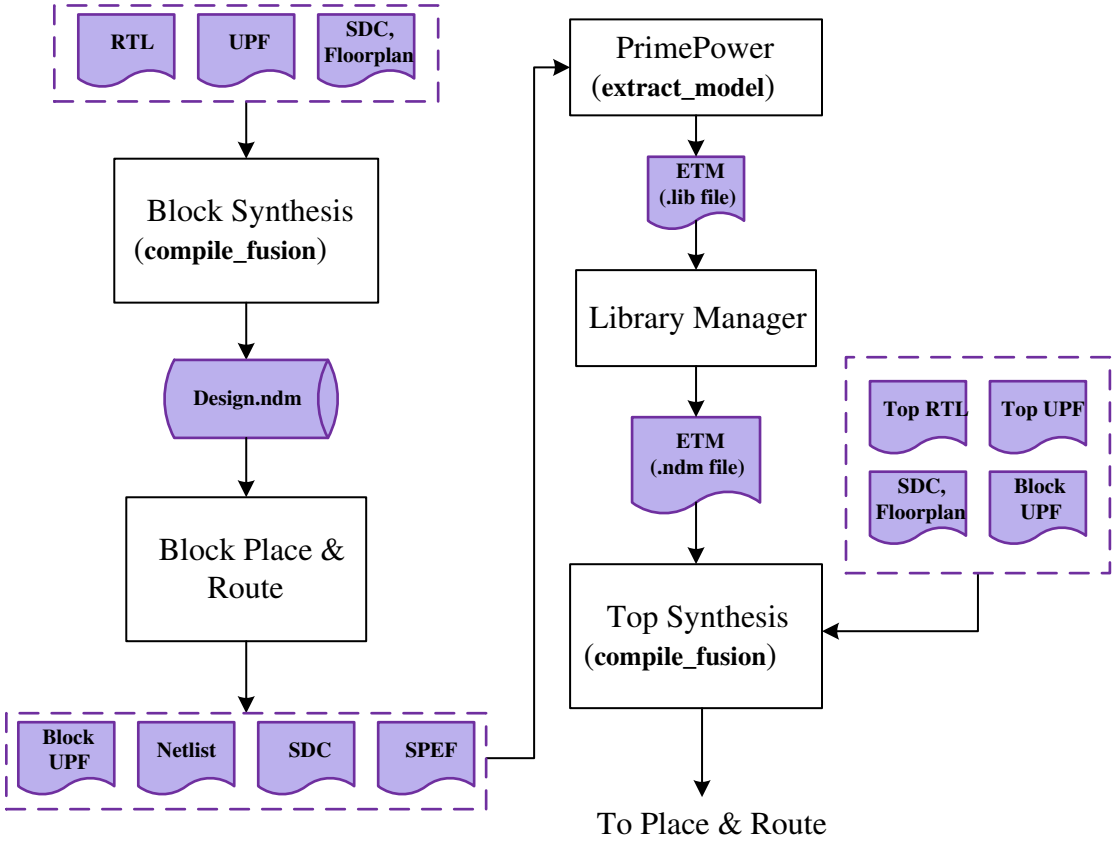
4. A `set_related_supply_net` command specified at the block-level scope
5. The power domain primary supply

The tool checks `set_port_attributes -driver_supply` commands at the top-level and block-level scopes for electrical violations only.

UPF files created with the `save_upf` command do not contain UPF constraints for ETMs, macros, or empty modules.

Figure 49 illustrates the hierarchical flow for ETMs and macros.

Figure 49 Hierarchical Synthesis Flow for ETMs and Macros

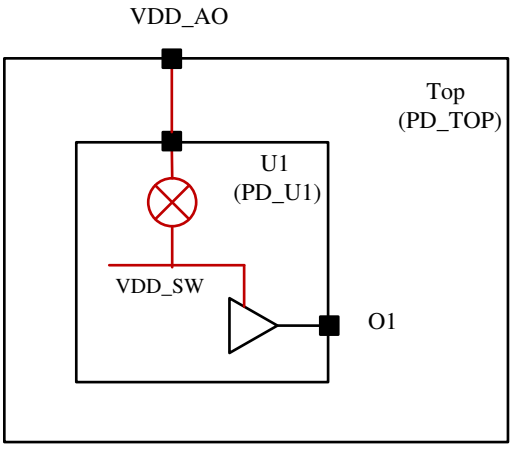


For the example in Figure 50, the U1 block is an ETM. You might have the following UPF for the top-level design:

```
create_power_domain PD_TOP -include_scope
create_supply_net VDD_AO
...
load_upf block.upf -scope U1
...
connect_supply_net VD_AO -ports U1/VDD_AO
```

Feedback

Figure 50 Example of an ETM in a Design



The block UPF might be as follows:

```
create_power_domain PD_BLK -include_scope
create_power_switch sw1 -domain PD_BLK \
    -input_supply_port {in VDD_AO} \
    -output_supply_port {out VDD_SW} ...
...
add_port_state sw1/out -state {ON 1.08} \
    -state {OFF off}
...
```

When you load the block UPF at the scope of the ETM, macro, or module, the UPF commands that refer to logic objects nested within the ETM or macro design are ignored.

Hard and Soft Macros

You can mark specific models as hard or soft macros for hierarchical implementation by using the `set_design_attributes` command. Use the `-models` option to specify the model names along with either the `-is_soft_macro true` or `-is_hard_macro true` option. When these options are specified, the tool defines either the `UPF_is_soft_macro` or `UPF_is_hard_macro` attribute for the specified models. If you specify the model list as {.}, the command applies to the model corresponding to the current scope.

You must specify the type of macro before loading the UPF for the macro. Alternatively, you can include the `set_design_attributes` command as the first command in the UPF file provided with the `load_upf -scope` command for the macro. For a hard macro, you can also specify the UPF with the `define_power_model` command.

A hard macro cell typically has a Liberty model that defines its interface, including supply ports and related supplies for its logic ports. A macro defined with the Liberty

`is_macro_cell` attribute is treated as a hard macro. A hard macro has one of the following:

- No UPF specification
- A self-contained UPF specification
- A UPF specification that does not define its own top-level domain

A soft macro always has a self-contained UPF. A hard macro might or might not have a UPF because it is not separately implemented by the tool.

For UPF processing, the tool performs checks for terminal boundaries on both hard and soft macros. The tool checks for a self-contained UPF for soft macros.

By default, the `find_objects` command considers all instances of hard and soft macros as leaf cells. The `-traverse_macros` option allows the command to traverse the macro terminal boundaries.

Setting Up Operating Conditions

Before you synthesize the design, you must consider the process, voltage, and temperature (PVT) operating conditions on the top level of the design hierarchy. The tool selects power management cells based on the cells available in the library and the PVT matching criteria that you specify.

Use the `opt.common.pvt_setting` application option to specify how the Fusion Compiler tool selects power management cells with respect to operating conditions. If used, this application option takes precedence over settings made with other commands or application options, such as the `opt.common.allow_incorrect_pvt` or `opt.common.prefer_exact_pvt` application options.

The `opt.common.pvt_setting` application option provides a unified method to control how the Fusion Compiler tool matches operating conditions when it selects power management cells from the library. To use this application option, set it to one of the valid values and optionally specify which power management cells to consider.

The valid values are as follows:

- `voltage_range_match`

  The default mode, in which the tool checks only for voltage range matching. For this check, the voltage specified by the `set_voltage` command must fall within the range of the minimum and maximum voltages of the library cell for each corner.

- `exact_match`

  For each timing corner, the tool checks the following criteria:

  - The process label of the library pane must match the `set_process_label` setting and the process number of the same library pane must match the same `set_process_number` setting.
  - For each supply net connected to the voltage rail of the library cell, the voltage of the rail in the same library pane must match the `set_voltage` setting of the supply net.
  - The temperature of the same library pane must match the `set_temperature` setting.

- `range_match`

  For each timing corner, the tool checks the following criteria:

  - The voltage specified by the `set_voltage` command must fall within the range of the minimum and maximum voltages of the library pane.
  - The temperature specified by the `set_temperature` command must fall within the range of the minimum and maximum temperatures of the library pane.

- `closest_match`

  For each timing corner, the tool chooses the closest match based on the following conditions, from highest to lowest priority:

  1. All PVT settings exact match
  2. Process label exact match
  3. Process number exact match
  4. Voltage exact match
  5. Voltage range match
  6. Temperature exact match

7. Temperature range match
8. All PVT settings mismatched

The identifiers for power management cells are as follows:

- `ls` (level-shifter cell)
- `iso` (isolation cell)
- `els` (enable level-shifter cell)
- `ret` (retention cell)
- `buf` (buffer)

To specify the power management cell type, append the application option setting with an equal sign (=) followed by the identifier. If you do not specify a cell type, the setting applies to all power management cells.

For example, the following command specifies exact PVT matching for level-shifter cells:

```
fc_shell> set_app_options -name opt.common.pvt_setting \
       -value "exact_match=ls"
```

You can combine multiple settings in one command. For example, the following command specifies exact PVT matching for level-shifter cells and range matching for isolation cells:

```
fc_shell> set_app_options -name opt.common.pvt_setting \
    -value "exact_match=ls, range_match=iso"
```

Reporting PVT Libraries

To display a report for a PVT for a specific corner case, use the `report_pvt` command. For the example in Figure 51, the report shows that the library saed32lvt has mismatches in the process number specified (1.1) and the effective process number (1). The asterisk to the left of the report shows which of the constraints (P, V, or T) are mismatched.

Feedback

Figure 51 Example of the report_pvt Output

```
****************************************
Report : pvt
Design :xxxx
Version: xxxx
Date   : xxxx
****************************************
--------------------------------------------------------------------------------
Warning: Corner corner_S1:  18 process number, 0 process label, 2 voltage, and 0 temperature mismatches. (PVT-030)
Warning: 130630 cells affected for early, 130630 for late. (PVT-031)
PVT-034: 116 port driving_cells affected for early, 116 for late.
--------------------------------------------------------------------------------
Warning: Mismatched process number in corner corner_S1. (PVT-022)
Library has 27 panes
Entry 161:
  Lib: saed32lvt
  Pane: 24
  Process Label:  specified: tt           effective: tt
* Process Number: specified: 1.1          effective: 1
  Rail 0 (default) Voltage: specified: --            effective: 1.05
  Rail 1 (VDD) Voltage: specified: 1.05         effective: 1.05
  Rail 2 (VDDG) Voltage: specified: --           effective: 1.05
  Rail 3 (VSS) Voltage: specified: 0.00         effective: --
  Primary rail: 1  Secondary rail: --  N-bias rail: --  P-bias rail: --
  Temperature:    specified: 125.00       effective: 125.00
  early cell refs: 71899    port refs: 0    driving_cell refs: 0    lib refs: 0
  late  cell refs: 0        port refs: 0    driving_cell refs: 0    lib refs: 0
```

Feedback

To examine the library, use the `report_lib` command.

Figure 52 report_lib Report

```
dcrt_shell> report_lib saed32lvt
****************************************
Report : library
Library: saed32lvt
Version: xxxx
Date   : xxxx
****************************************
Power rails:
Index  Name         Type
  0    <default>    power
  1    VDD          power
  2    VDDG         power
  3    VSS          ground
Pane count: 4
Pane 0:
  Process label: (none)
  Process number: 1.01
  Voltage rail count: 3
  Voltage for rail 0 (<default>): 0.95
  Voltage for rail 1 (VDD): 0.95
  Voltage for rail 2 (VDDG): 0.95
  Temperature: 125
```

4

Lower-Domain Boundary Support

You can control many aspects of power management cells insertion with respect to power domain boundaries.

For more information, see the following topics:

- Overview of Power Domain Boundaries
- Applying Isolation, Level-Shifter, and Repeater Strategies

Overview of Power Domain Boundaries

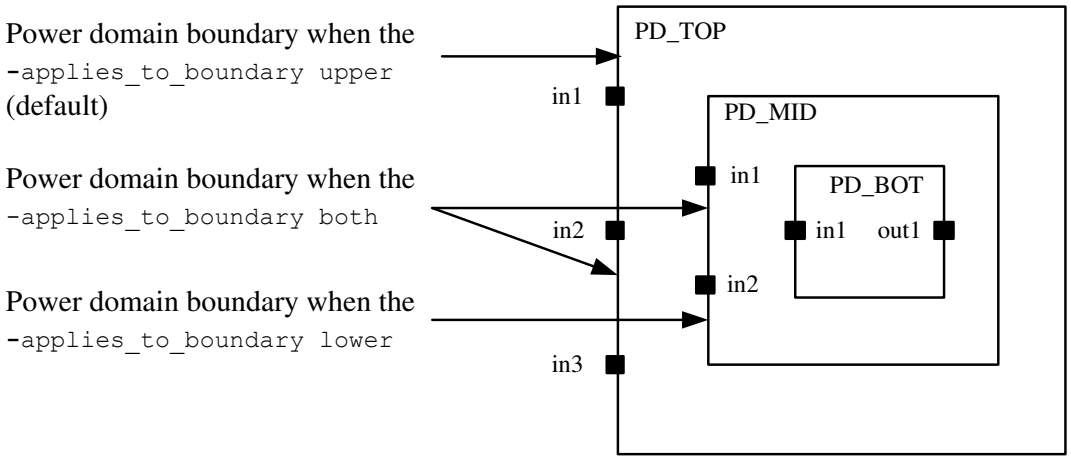
By default, the Fusion Compiler tool considers the logical boundary of the root cells of the power domain as the boundary of the power domain. However, to comply with the IEEE 1801 (UPF) standard, the tool can consider a power domain boundary to include the boundary of another domain contained in it. This feature enables you to specify the elements on the lower-domain boundary for level-shifter and isolation strategy definition, which gives you additional flexibility in selecting the location of the power management cells.

To extend the definition of the power domain boundary to the boundary of another power domain contained in it, use the `-applies_to_boundary` option in the `set_isolation` and `set_level_shifter` commands as shown in the following example:

```
set_isolation ISO1 -domain PD_TOP -applies_to_boundary lower
```

Feedback

Figure 53 Definition of Power Domain Boundaries for PD_TOP



By default in Figure 53, the tool considers only in1, in2, and in3 ports of the PD_TOP domain to be at the domain boundary. This is also the case if the `-applies_to_boundary` option is set to `upper`.

When the `-applies_to_boundary` option is set to `both`, the tool considers the in1, in2, in3, MID/in1, and MID/in2 ports to be at the power domain boundary. However, the boundary does not extend to the interface of the BOT design or the PD_BOT power domain.

When the `-applies_to_boundary` option is set to `lower`, the tool considers the MID/in1 and MID/in2 ports to be at the power domain boundary.

Note that the boundary does not extend to the interface of the BOT design or the PD_BOT domain.

To set the lower boundary of a power domain at the HighConn side of all hard macros included within the power domain, you must set the `macro_as_domain_boundary` design attribute to `true`. This is a scope-level attribute that indicates whether macros at or below that scope need to be considered as design boundaries. Use the `-elements` option to apply the attribute to specific elements. In addition, you must execute one of the following actions:

- Set the design attribute `lower_domain_boundary` to `true` for the top-level scope or a block-level scope.
- Specify the `-applies_to_boundary lower` or `-applies_to_boundary both` option of the `set_isolation`, `set_level_shifter`, or `set_repeater` commands.

In the following example, the lower domain boundary of power domain PD_TOP includes the ports of all macros, with the exception of macros contained in cell U2. As a result, the tool inserts isolation cells at the input ports of all macros except macros contained in cell U2.

Feedback

```
set_design_attributes -elements {.} \
        -attribute macro_as_domain_boundary true
set_design_attributes -elements {U2} \
        -attribute macro_as_domain_boundary false
create_power_domain PD_TOP -elements {.}
set_isolation ISO -domain PD_TOP -applies_to outputs \
        -applies_to_boundary both
```

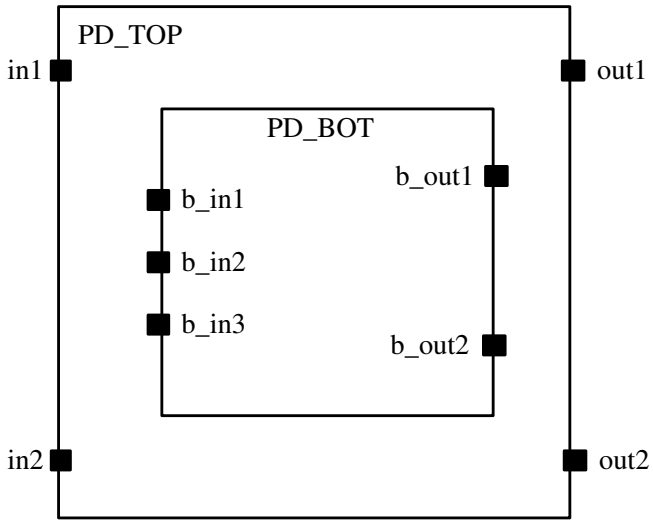
When the `macro_as_domain_boundary` attribute is set to `true` for specific hard macros, the `terminal_boundary` attribute is allowed on the pins of the specified macros. In addition, you can specify pins or instances of the macros by using the `-clamp_value` and `-repeater_supply` options of the `set_port_attributes` command, or by setting the `repeater_power_net` and `repeater_ground_net` attributes with the same command.

Applying Isolation, Level-Shifter, and Repeater Strategies

When you define the level-shifter, isolation, and repeater strategies with the `-applies_to_boundary` option, you specify the domain and domain boundary to which the strategy applies. The strategy applies to the pins of the domain boundary specified by the `-applies_to_boundary` option.

For example, in the nested power domains shown in Figure 54, the b_out1 and b_out2 output pins of the PD_BOT power domain are input pins for the strategies defined in the PD_TOP power domain. Similarly, b_in1, b_in2, and b_in3 input pins of the PD_BOT power domain are the output pins for the strategies defined in the PD_TOP power domain.

Figure 54 Example of Nested Power Domains



The strategies that apply to the input pins of the power domain also apply to the output pins of the root cell of the power domain nested inside the power domain. Similarly, the

strategies that apply to the output pins of the power domain also apply to the input pins of the root cells of the power domains nested inside the domain.

For Figure 54, the following table shows which pins apply to the specified boundary crossing for a strategy defined at PD_TOP.

Table 20 Pins Considered for Boundary Crossings

-applies_to_boundary	Pins
upper	in1, in2, out1, out2
lower	b_in1, b_in2, b_in3, b_out1, b_out2
both	in1, in2, out1, out2, b_in1, b_in2, b_in3, b_out1, b_out2

Specifying Domain Boundaries With the -applies_to Option

You can use the `-applies_to` option of the `set_isolation` , `set_level_shifter`, and `set_repeater` commands to filter strategies to specific pins. For the nested domains in Figure 54, the example in Example 12 illustrates the lower-domain boundary with the `-applies_to` option.

Example 12 Lower-Domain Boundary Specification

```
create_power_domain PD_TOP -include_scope
set_isolation out_iso -domain PD_TOP -applies_to output \
                      -applies_to_boundary lower
```

In Example 12, the out_iso strategy defined for the PD_TOP power domain applies to the b_in1, b_in2, and b_in3 pins, which are the lower-domain boundary output pins of the PD_TOP power domain.

Example 13 Upper-Domain Boundary Specification

```
create_power_domain PD_TOP -include_scope
set_isolation both_iso -domain PD_TOP -applies_to output \
                       -applies_to_boundary upper
```

In Example 13, the boundary specification is `both` and both_iso strategy for PD_TOP applies to out1 and out2, which are the upper-domain boundary output pins.

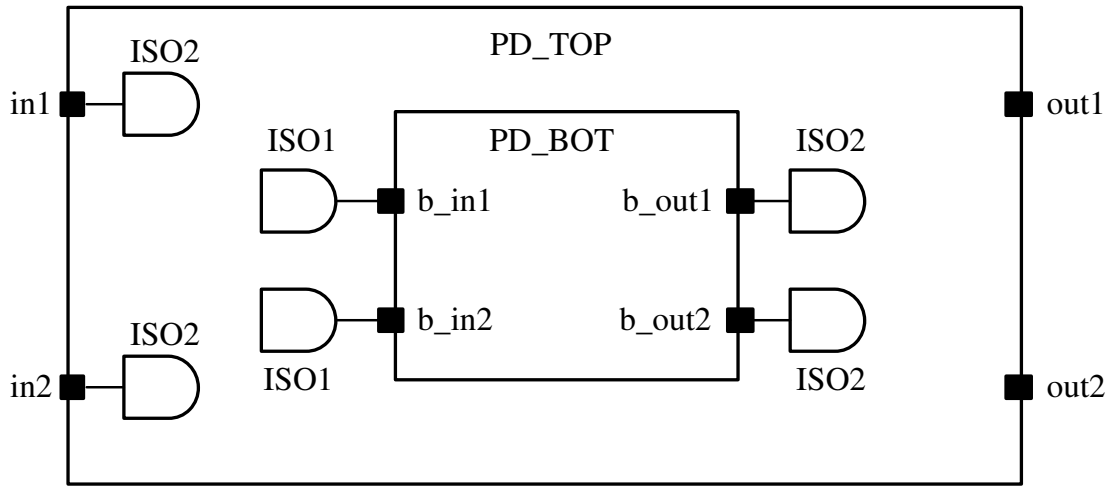
Defining Cell Placement With the -location Option

When you define a strategy using the `set_isolation` or `set_level_shifter` commands, the tool supports both the `-location parent` or `-location self` options. You can use the location placement along with the `-applies_to` option for added flexibility in placing your isolation or level-shifter cells.

Feedback

Figure 55 illustrates how the tool applies the isolation strategies for Example 14.

Figure 55 Isolation Cell Insertion With the Domain Boundary Specified

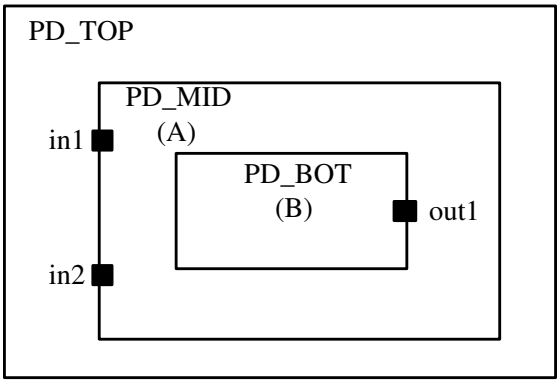


Example 14 Strategies With Different Port Directions

```
create_power_domain PD_TOP
create_power_domain PD_BOT
set_isolation ISO1 -domain PD_TOP -applies_to output \
                   -applies_to_boundary lower
set_isolation ISO2 -domain PD_TOP -applies_to input \
                   -applies_to_boundary both
```

For the example in Figure 55, the isolation cells are placed in domain PD_TOP because the strategies are defined for domain PD_TOP and the `-location` option is not specified. By default, the location is `self`.

Figure 56 Isolation Cell Insertion With the Parent Location Specified



For Figure 56, if you have the following:

```
create_power_domain PD_TOP
create_power_domain PD_MID -elements {A}
create_power_domain PD_BOT -elements {A/B}
set_isolation ISO1 -domain PD_MID -applies_to inputs \
                   -applies_to_boundary both -location self
```

The tool applies ISO1 strategy to ports A/in1, A/in2, and A/B/out1.

Suppose for Figure 56, you have the following:

```
create_power_domain PD_TOP
create_power_domain PD_MID -elements {A}
create_power_domain PD_BOT -elements {A/B}
set_isolation ISO1 -domain PD_MID -applies_to inputs \
                   -applies_to_boundary both -location parent
```

The tool implements ISO1 only on A/in1 and A/in2 at the parent location PD_TOP. However, the tool does not apply the strategy to port B/out1 because the parent domain for PD_MID is PD_TOP and the tool cannot place the isolation cell there. In this case, the tool issues a warning message.

5

Always-On Logic

Generally, multivoltage designs have some power domains (shutdown domains) that are shut down and powered up during the operation of the chip, while other power domains remain powered up at all times (always-on domains). The control nets that connect cells in an always-on power domain to cells within a shutdown domain must remain on during shutdown. These paths are referred to as always-on paths.

This section covers the following topics:

Always-On Design

Shutdown domains usually contain objects that must stay active, such as retention registers, isolation cells, retention control paths, and isolation enable paths. These objects are referred to as always-on. 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.

Figure 57 shows an example of a feedthrough net that crosses a shutdown domain. Figure 58 shows an example of control signals for a retention register, which must always be active. Control signals for isolation cells must also remain active at all times.

Feedback

Figure 57 Feedthrough Net Crossing a Shutdown Voltage Region

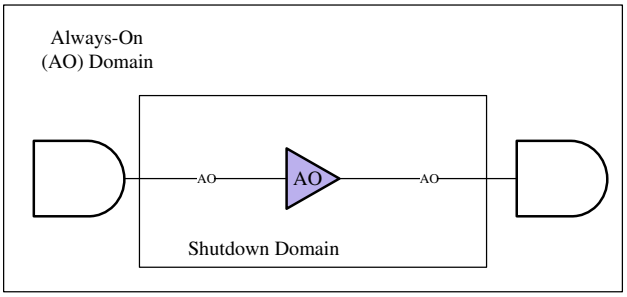
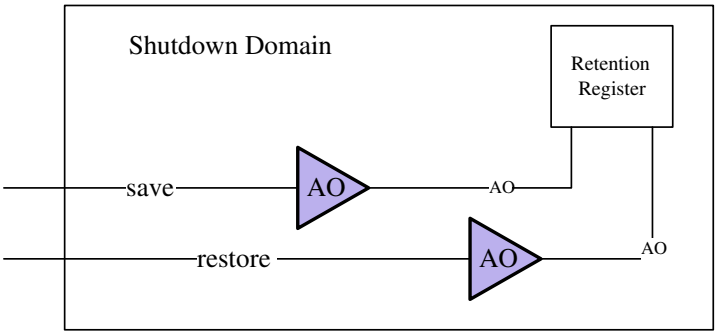


Figure 58 Control Signals for a Retention Register

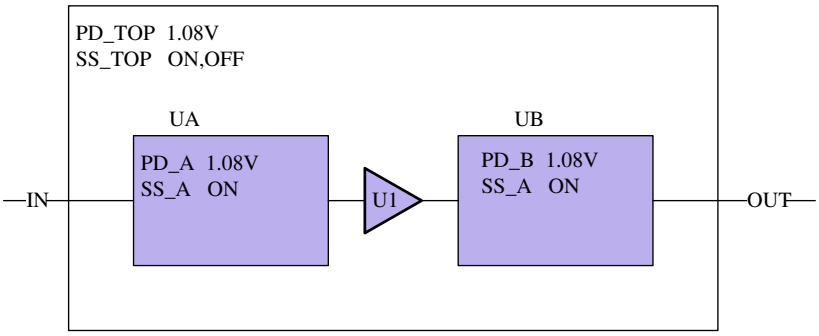


Always-on optimization occurs automatically during synthesis. The tool uses dual-rail or single-rail buffers, depending on the cell availability and physical constraints.

Always-On Bias Supplies

During always-on synthesis, the tool must choose which supply to use to power the inserted always-on buffers. For example, in Figure 59, buffer U1 is on a feedthrough net in a domain that powers down, but the buffer must remain on.

Figure 59 Always-On Buffer Using a Bias Supply



Feedback

The tool uses the supply of either the load pin or the driver pin to power the buffer because those supplies are always on. If both the driver and load supplies are not available in the feedthrough domain, the net is marked with the `dont_touch` attribute. The power and ground supplies of the buffer are connected to the SS_A.power and SS_A.ground supplies. The backup power pin of buffer U1 uses the bias supply from PD_TOP.primary.nwell (or PD_TOP.primary.pwell) if the bias supply is more on or equally on compared to SS_A.

In general, for an always-on cell, the tool tries to use the bias supplies of the local domain for the buffer's backup pin, provided that the bias supply is more on or equally on compared to the corresponding driver or load power supply.

Always-On Cell Attributes

The Fusion Compiler tool sets the following cell attributes for always-on cells:

- The `is_always_on_logic` attribute is `true` for any tie cell, buffer cell, or inverter cell that has an input or output power or ground supply that is different from the corresponding primary supply of the power domain where the cell is located.
- The `is_user_always_on_logic` attribute is `true` for any cell marked with the `is_always_on_logic` attribute that has a user-generated exception connection.

You can use the `get_cells` command with these attributes to query the always-on cells in a design. For example, the following command finds all of the always-on cells:

```
fc_shell> get_cells -filter is_always_on_logic==true -hierarchical
```

The following command generates a collection of all single-rail always-on cells:

```
fc_shell> get_cells -filter "is_always_on_logic \
   && !has_multi_power_rails"
```

Voltage-Aware Always-On Synthesis

You can enable voltage-aware always-on synthesis on certain physical feedthrough paths. This enables buffer insertion on feedthrough paths when there is a disjoint voltage area even though, logically, the voltage areas belong to the same hierarchy.

This topic contains the following:

Feedback

- Feedthrough Buffering With Missing Voltage Areas
- Analyzing Feedthrough Buffering

Feedthrough Buffering

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

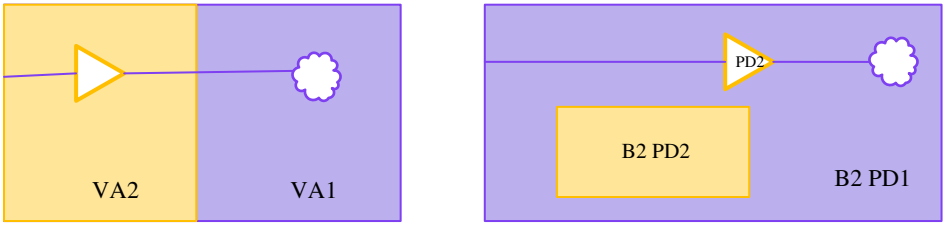
By default, the tool inserts regular buffers on always-on feedthrough nets, instead of always-on buffers, if the primary supply of the feedthrough voltage area is more on than the sink supply and the primary supply is off when the source supply is off.

Physical Feedthrough Buffering

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

For example, in Figure 60, the feedthrough net physically crosses the VA2 voltage area (the PD2 power domain) while logically, the net and the buffer reside in the VA1 voltage area (the PD1 power domain). The tool assigns the buffer to the PD2 power domain even though it resides in a different logic hierarchy. When the tool writes the output UPF, the buffer is added to the elements list of the `create_power_domain` command of the PD2 domain.

Figure 60 Example of Physical Feedthrough Buffering



Physical View

Logical View with Physical Feedthrough Buffer

Logical Feedthrough Buffering

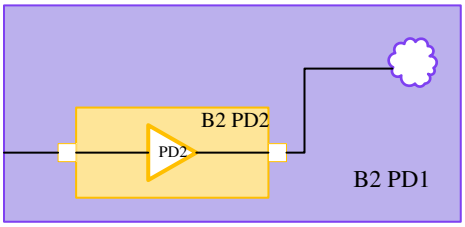
In some cases, the tool cannot implement physical feedthrough buffering if the power domain in which the buffer resides is created at a scope below the net's logical hierarchy.

Feedback

For example, in Figure 61, if the PD2 power domain is created at a scope B2 and it is below the net's logical hierarchy, the buffer cannot be logically placed in the PD1 domain because it belongs to the PD2 domain. The PD1 domain is not visible to the PD2 domain.

In this case, the tool implements a logical feedthrough buffer in which the buffer is placed in the PD2 domain. The tool punches ports to enable the net to traverse the PD2 domain. No isolation cells can be inserted at the punched ports.

Figure 61 Example of Logical Feedthrough Buffering



Logical View with Logical Feedthrough Buffer

Enabling Logical and Physical Feedthrough Buffering

To enable physical feedthrough buffering, use the `create_voltage_area_rule` command and set the `-allow_physical_feedthrough` option to `true`.

To enable logical feedthrough buffering, use the `create_voltage_area_rule` command and set the `-allow_logical_feedthrough` option to `true`.

In addition, you can use the following options to control logical feedthrough buffer insertion:

- The `-include_logical_feedthrough_hierarchy` option specifies the hierarchies in which the tool can insert logical feedthrough buffers.
- The `-exclude_logical_feedthrough_hierarchy` option specifies the hierarchies in which the tool cannot insert logical feedthrough buffers.

These options are mutually exclusive and can only be used if the `-allow_logical_feedthrough` option is set to `true`.

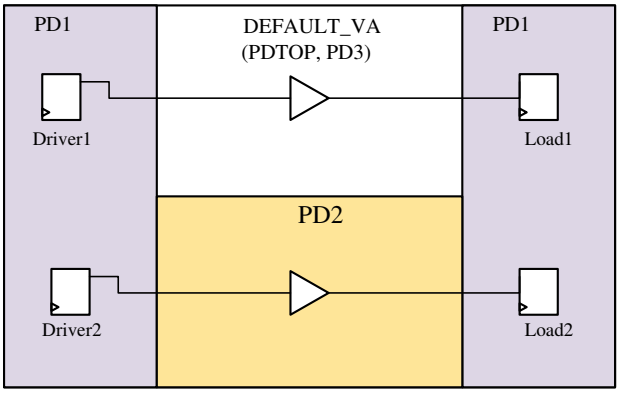
When using logical feedthrough buffering for physical synthesis, you should run the `create_voltage_area_rule` command before you run the `commit_upf` or `create_mv_cells` commands. For logical synthesis, the `commit_upf` command is performed automatically as part of the `compile_fusion` command.

Feedback

Feedthrough Buffering With Missing Voltage Areas

If your design has missing voltage areas, the tool allows physical or logical feedthrough buffering through the domain if the domain's primary supply is equivalent to the topmost domain's primary supply.

Figure 62 Feedthrough Buffering With a Missing Voltage Area



In Figure 62, the buffer can be inserted in the PD3 domain and placed in the DEFAULT_VA voltage area because PDTOP and PD3 have the same primary supply. The buffer can also be inserted in PD2 because that domain has a defined voltage area.

Analyzing Feedthrough Buffering

During design planning, the `place_pins` command creates pins (feedthrough ports) for feedthrough nets that pass through cells. Depending on the supply constraints for the feedthrough nets and the cells they pass through, multivoltage violations might occur.

The Fusion Compiler tool can analyze feedthroughs and provide feedback about the design. The feedback includes a description of the violation and suggested changes to the isolation and level shifter strategies to fix the violations. You must make the fixes to the netlist manually.

Use the `generate_mv_feedback` command to perform this analysis. You must use one of the `-net` or `-pin` options to specify a list of nets or pins. Feedback is provided for feedthrough pins connected to the specified nets or pins. By default, the tool provides feedback that includes both isolation and level shifter strategies. However, you can restrict the feedback to one type by using either the `-isolation` or `-level_shifter` option.

The Fusion Compiler tool evaluates the primary supply and the available supplies of the feedthrough power domains. In addition, the tool checks the relative amount of time that related supplies are in the ON state for the blocks. The tool provides a buffering recommendation for each multivoltage violation based on evaluating costs. The tool does not evaluate the library cell availability for buffer, isolation, and level-shifter cells.

The cost considerations are as follows, from lowest cost to highest cost:

- A single-rail buffer that uses the primary supply without introducing a multivoltage violation
- A dual-rail buffer that uses available supplies without introducing a multivoltage violation
- An isolation or level-shifter cell and a single-rail buffer that use the primary supply without introducing a functional issue; isolation or level-shifter violations might occur
- An isolation or level-shifter cell and a single-rail buffer that use the primary supply, but introducing a functional issue; isolation or level-shifter violations might occur

A functional issue occurs if the feedthrough path goes through cells with supplies that might shut down even when the feedthrough driver and load are powered on.

The types of feedback and the keywords used to describe them in reports are shown in Table 21.

Table 21 Feedback Types for Multivoltage Violations

Feedback	Keyword
Isolation strategy that should be moved from one block to another to avoid redundant isolation strategies	`move_iso_strategy`
Level-shifter strategy that should be moved from one block to another to avoid redundant level-shifter strategies	`move_ls_strategy`
Isolation strategy that needs to be newly created in a block to resolve a static multivoltage violation caused by feedthrough buffering	`create_iso_strategy`
Level-shifter strategy that needs to be newly created in a block to resolve a static multivoltage violation caused by feedthrough buffering	`create_ls_strategy`
Isolation strategy that needs to be newly created in a block to resolve a static multivoltage violation caused by feedthrough buffering, but resulting in a functional issue	`create_iso_strategy_ with_func_viol`
Wrong isolation enable caused by feedthrough buffering	`wrong_iso_enable`

You can save the feedback to a file by using the `-output` option of the `generate_mv_feedback` command.

Feedback

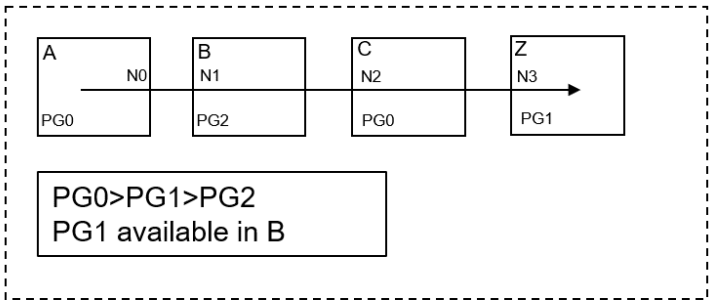
For each net or pin specified in the `generate_mv_feedback` command, the report includes one or more strategy recommendations. Each type of feedback is reported in a slightly different format. For example, feedback for the `move_iso_strategy` type is as follows:

```
Begin_driver_pin: pin_name1
   FeedbackType: move_iso_strategy
      from_pd_boundary_pin: pin_name2
      to_pd_boundary_pin: pin_name3
      Targets of to_pd_boundary_pin: list_of_pins
End_driver_pin: pin_name1
```

The report also includes the related supply information on the feedthrough block.

For example, consider the feedthrough in Figure 63.

Figure 63 Feedthrough With Related Supply on Feedthrough Ports



The report from the `generate_mv_feedback` command is as follows:

```
Begin_driver_pin: A/N0
   FeedbackType: create_iso_strategy
      at_pd_boundary_pin: C/N2
      Target of pd_boundary_pin: Z/N3
      Target of driver_pin: Z/N3
End_driver_pin: A/N0
```

The buffering solution is:

```
A(PG0)->B(PG1)->C(PG0)->Z(PG1)
```

In block B, the available supply PG1 is used for buffering because using the primary supply PG2 causes a functional issue, which has a higher cost than using the available supply.

Feedback

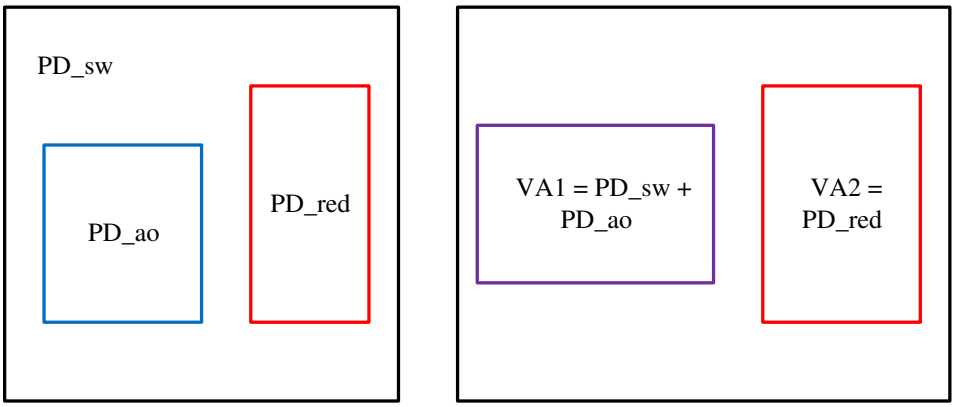
Always-On Block Synthesis

In some situations, you might want to merge power domains with different power state table states to a common voltage area. This enables you to synthesize a block design and share its voltage area with another power domain using a different primary supply.

To do this, the following conditions must be met:

- The power grid in the common voltage area match the primary supply of at least one of the power domains
- The other domains are implemented using dual-rail cells

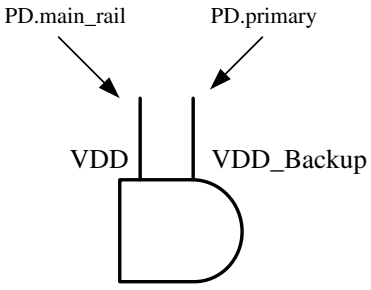
Figure 64 Merging Power Domains to a Common Voltage Area



For the example in Figure 64, some cells in PD_ao reside in the same voltage area as PD_sw. The power grid of this voltage area, VA1, matches the primary supply of PD_sw.

When you use the `main_rail` supply handle, the `primary_power` PG pin of the cell is implicitly connected to the `main_rail` supply.

Figure 65 Supply Connections for Dual-Rail Cell



Feedback

Dual-rail library cells must be included in your reference libraries for the tool to use during mapping and optimization.

Implementing Always-On Blocks

To merge domains to a common voltage area in Figure 64, domain PD_ao is synthesized using dual-rail cells. Use the supply handle named `main_rail` to model the supply that matches the power grid of this common voltage area.

Domains that you want to merge to a common voltage area should have the same `main_rail` supply. For domains that do not specify the `main_rail` as a supply, the `main_rail` supply is assumed to be the domain's primary supply. For example:

```
create_supply_set SS_sw
create_supply_set SS_ao
create_power_domain PD_ao -supply {primary SS_ao} \
                          -supply {main_rail SS_sw}
create_power_domain PD_sw -supply {primary SS_sw}
```

The following commands associate domains PD_sw and PD_ao to the same voltage area:

```
create_voltage_area -power_domains PD_sw
set_voltage_area -add_power_domains {PD_ao}
```

By default, the tool does not capture the supply net connections for the backup power PG pin that is implicitly connected to the domain's primary supply. To force the tool to write out these supply net connections to the output UPF, set the `mv.upf.write_csn_for_dual_rail_cells_in_ao_domain` application option to `true`.

Specifying Secondary PG Constraints

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

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

Secondary PG constraints help the tool to identify the location of straps that have limited physical availability in the voltage area. The tool honors these constraints throughout the flow. Dual-rail buffers affected by these constraints are placed in areas with secondary power straps. Dual-rail cells have priority over single-rail cells for placement in these areas.

Feedback

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

The advanced legalizer is required to support secondary PG constraints. For advanced technology nodes, the advanced legalizer is enabled when you set the technology node, as follows:

```
fc_shell> set_technology -node node_number
```

Alternatively, if you know that the advanced legalizer supports a specific older technology node, set the following application option:

```
fc_shell> set_app_options -name place.legalize.enable_advanced_legalizer\
          -value true
```

Topics covered in this section:

- Creating Secondary PG Constraints Manually
- Creating Secondary PG Constraints Automatically
- Checking Secondary PG Constraints and Resolving Conflicts
- Hierarchical Secondary PG Placement Constraints

Creating Secondary PG Constraints Manually

Use the `create_secondary_pg_placement_constraints` command to define secondary PG constraints manually. When you use this command, you must specify one of the following options:

- `-layers`

  The tool determines the secondary power supply regions based on the route shapes of the specified supply and layers.

- `-region`

  You specify the regions where the secondary power straps are available.

After you create all of the secondary PG placement constraints, apply them by using the `commit_secondary_pg_placement_constraints` command. If you change secondary PG placement constraints, the floorplan, or the UPF, you must run the `commit_secondary_pg_placement_constraints` command again.

To remove constraints, use the `remove_secondary_pg_placement_constraints` command.

Feedback

Making a Supply Unavailable

Even when a secondary supply is available in the UPF, you might later discover that there are not enough routing resources to create the PG straps. In this case, you can make the secondary supply physically unavailable in portions of the design, which overrides the UPF. Physically unavailable supplies are not used during buffering optimization.

You cannot make a supply unavailable if it is

- The primary supply in the voltage area
- A secondary supply that is used by an existing cell or port in the voltage area
- A secondary supply that is specified for isolation, level shifter, retention, or repeater UPF strategies in the power domain that corresponds to the voltage area

To make a supply physically unavailable, use the `-exclude_supply` option of the `create_secondary_pg_placement_constraints` command.

If voltage areas are specified with the `-exclude_supply` option, the supply is unavailable only in the specified voltage areas. The `-exclude_supply` option does not affect layers or regions.

For example, suppose your UPF file has the following supplies:

```
create_power_domain PD1 -available_supplies {secpg0 secpg1} ...
```

Later, if only secpg0 is available and secpg1 is not available, use the following commands:

```
create_voltage_area -name VA1 -power_domains PD1 ...
create_secondary_pg_placement_constraints -name secpg0 \
      -supply secpg0 -voltage_areas VA1 ...
create_secondary_pg_placement_constraints -name secpg1 \
      -exclude_supply secpg1 -voltage_areas VA1 ...
```

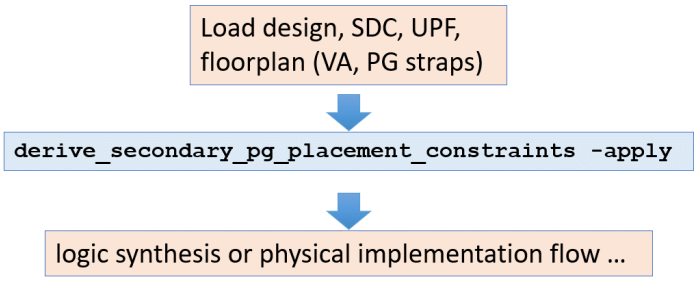
Creating Secondary PG Constraints Automatically

Use the `derive_secondary_pg_placement_constraints` command to generate secondary PG constraints automatically. The Fusion Compiler tool analyzes the secondary PG straps and generates a set of `create_secondary_pg_placement_constraints` commands with the appropriate `-supply` or `-exclude_supply` options.

You can use the `derive_secondary_pg_placement_constraints` command in a single step, as shown in Figure 66. The `-apply` option automatically commits the constraints.

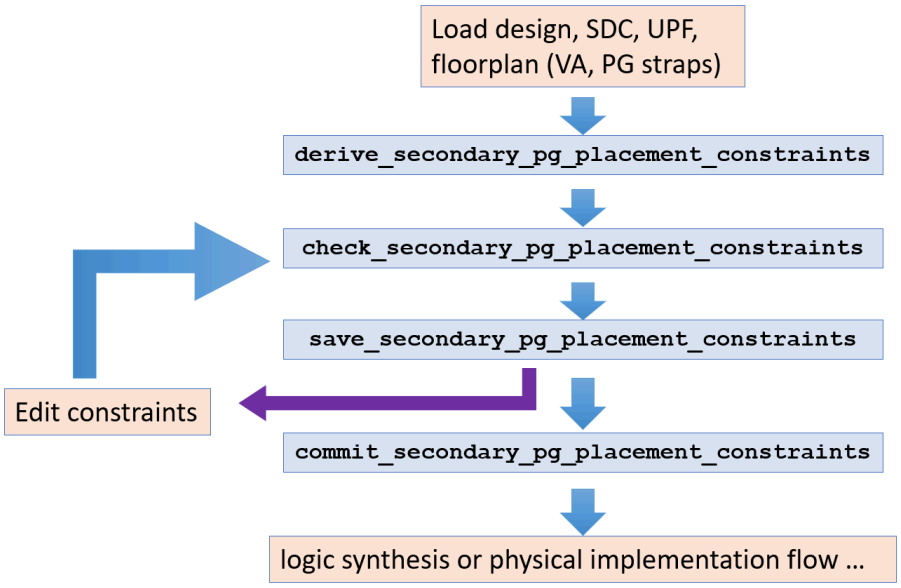
Feedback

Figure 66 Automatic Creation of Secondary PG Constraints



You can also use the `derive_secondary_pg_placement_constraints` command interactively in combination with manual constraint specifications, as shown in Figure 67. This flow allows you to use the automatic constraints as a starting point, then refine or add constraints as needed.

Figure 67 Interactive Flow for Secondary PG Constraint Creation



Feedback

Usage notes for the `derive_secondary_pg_placement_constraints` command are as follows:

- You must use the `-layers` option to specify the layers where straps are physically available. The Fusion Compiler tool analyzes the secondary PG straps and generates a set of `create_secondary_pg_placement_constraints` commands with the `-supply` option for the supplies with straps available in the specified layers.
- If PG straps are not found on the specified layers for supplies listed in the UPF, the tool generates `create_secondary_pg_placement_constraints` commands with the `-exclude_supply` option for those supplies.
- You can optionally use the `-margin` option to define the extent of secondary PG voltage area shapes based on the pitch of the secondary PG straps.

Voltage area shapes created from the secondary PG placement constraints have the following attributes:

- The `is_secondary_pg_region` attribute is set to `true` if the constraint is created by secondary PG placement constraints.
- The `secondary_pg_nets` attribute specifies the names of the secondary nets.

Checking Secondary PG Constraints and Resolving Conflicts

Use the following commands to check the secondary PG routing constraints:

- `check_secondary_pg_placement_constraints`

  Checks for conflicts between secondary PG constraints and the UPF or netlist, between user-specified secondary PG constraints and the strap availability, and between user-specified and tool-derived secondary PG constraints.

- `report_secondary_pg_placement_constraints`

  Reports both committed and uncommitted constraints and indicates if the constraint is user-specified or tool-derived.

- `save_secondary_pg_placement_constraints`

  Writes an ASCII file that contains all user-specified and tool-derived secondary PG constraints.

- `check_legality`

  Checks the cell placement and identifies cells that violate placement constraints.

- `check_bufferability`

  Determines whether the voltage area shape has straps available for buffer insertion.

Feedback

The `commit_secondary_pg_placement_constraints` command does not require that you resolve all conflicts before executing the command. Instead, the tool resolves conflicts as shown in Table 22.

Table 22 Automatic Conflict Resolution

Condition	Action During Commit
An excluded supply in a constraint is available in the UPF	The excluded supply is considered to be physically unavailable
An excluded supply in a constraint is used in a power management cell strategy in the UPF	The tool ignores the constraint and issues a warning
An excluded supply in a constraint is connected to a leaf pin or top-level port in the netlist	The tool ignores the constraint and issues a warning
User-specified constraints specify the same supply with both `-supply` and `-exclude_supply` options	The tool honors the constraint with the `-supply` option and issues a warning
A user-specified constraint with the `-exclude_supply` option conflicts with a tool-derived constraint with the `-supply` option	The tool honors the user-specified constraint and issues a warning
A tool-derived constraint with the `-exclude_supply` option conflicts with a user-specified constraint with the `-supply` option	The tool honors the user-specified constraint and issues a warning
Multiple user-specified constraints with the `-supply` option exist for the same supply	The tool creates simplified voltage area shapes to cover the union of the user-specified shapes
A user-specified constraint with the `-supply` option conflicts with a tool-derived constraint with the `-supply` option and layer specifications	The tool honors the user-specified constraint and issues a warning

Hierarchical Secondary PG Placement Constraints

You can use secondary PG constraints during design planning to define where secondary supplies are available within the subblocks of a design. The general flow is as follows:

1. Use the `split_constraints` and `commit_block` commands to create the subblocks.
2. Create voltage area shapes by using the `shape_blocks` command.
3. Create PG straps at the top level, then use the `characterize_block_pg` command to create the PG strategies for the subblocks.

4. Create the PG straps at the block level by using the `compile_pg` command.
5. Create secondary PG constraints for each block and commit the constraints.
6. Create secondary PG constraints at the top level and commit the constraints.

When you commit the constraints at the top level of the design, use the `commit_secondary_pg_placement_constraints -commit_subblocks` command. The `-commit_subblocks` option checks all block-level secondary PG constraints and commits any uncommitted constraints. The tool then creates voltage area shapes based on the secondary PG constraints and keeps track of supplies available in or excluded from each voltage area shape.

To list the constraints, use the `report_secondary_pg_placement_constraints -all_blocks` command. To check for consistency and error conditions in specific blocks, use the `check_secondary_pg_placement_constraints -blocks` command with a list of blocks.

For more information, see the *Fusion Compiler Design Planning User Guide*.

Always-On Cells Without Primary PG Pins

Some types of always-on cells have backup supply PG pins, but do not have primary supply PG pins. These cells are single-rail cells that rely on the backup supply, not the primary supply. Such cells can provide always-on functionality without the complexities of dual-rail cells.

These always-on cells connect directly to the secondary supply straps by vias (pins) and do not require secondary PG routing. Any number of these always-on cells can be placed in an abutting arrangement to provide continuity of the secondary supply. The primary supply might physically traverse the always-on cells for continuity with other cells, but the primary supply pins are not exposed logically in the cells.

This design style requires n-well and p-well isolation between the always-on cells and normal cells. Therefore the library must also contain specific cells (well breaker cells) that interrupt the well continuity and dictate the spacing of normal cells with respect to the always-on cells.

The Fusion Compiler tool supports isolation cells and repeater cells that have backup supply PG pins and no primary supply PG pins. The library cells must have the following properties:

- The `always_on` attribute is `true`.
- The `is_ao_without_primary_pg_pin` attribute is `true`.

- The cell has a backup power pin.
- The cell has no primary power pin.

To use these types of cells in the Fusion Compiler tool, set the `mv.cells.ao_cells_without_primary_pg_pin` application option to `true`. The tool inserts the cells as needed and applies `connect_supply_net` statements for the secondary supply in the derived section of the UPF. You must place the well breaker cells manually, but the legalizer accepts them.

For example, Figure 68 illustrates three types of cells: a normal cell with pins to the primary power supply, a special always-on cell with pins only to the secondary power supply, and a well breaker cell. Figure 69 illustrates how several always-on cells can be abutted to provide continuity for both the primary and secondary supplies, but with a contact only to the secondary power supply. In addition, a well breaker cell is placed at both ends of the row of always-on cells.

Figure 68 Always-On Cells

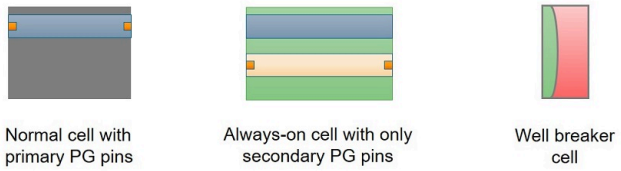
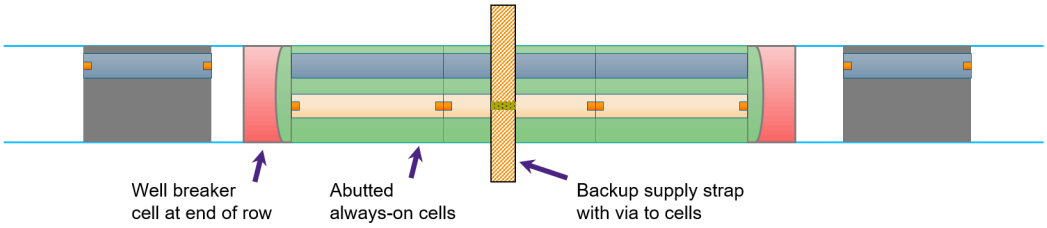


Figure 69 Always-On Cells in the Layout



The `fix_mv_design -buffer` command can replace preexisting dual-rail repeaters with secondary-supply-only repeaters. However, bias designs are not supported.

Always-On Legalization of Preinstantiated RTL Buffers and Inverters

The Fusion Compiler tool supports the always-on (AO) legalization of preinstantiated buffers and inverters in the RTL netlist, in `create_mv_cells` and `compile_fusion`.

Feedback

This AO legalization step is equivalent to running `fix_mv_design -buffer` before or after `compile_fusion`, and supports all the optimizations currently supported by `fix_mv_design -buffer`.

- In compile_fusion
- In create_mv_cells

In compile_fusion

During multivoltage cell insertion, to enable AO legalization of preinstantiated RTL buffers and inverters, set the application option `mv.upf.ao_legalize_gtech_buf` option to true, before running the `compile_fusion` command.

```
set_app_options -as_user_default
   -name mv.upf.ao_legalize_gtech_buf -value true
```

When the option is enabled, during `compile_fusion`, the tool performs the following steps:

- Runs the AO legalization of buffers/inverters after isolation cell insertion, so that the AO legalization will not affect isolation source/sink strategies.
- Runs the AO legalization at the same time as level-shifter insertion. This means greater flexibility to insert a new level shifter, convert a single-rail GTECH buffer/inverter into dual-rail, or do both. This flexibility will help to fix voltage violations which could not be solved previously because buffers/inverters act as hard terminals with fixed supplies. The flexibility allows to change supplies to these buffers/inverters by converting them into dual-rail.
- Converts single-rail buffers/inverters into dual-rail user lib cells, not dual-rail GTECH lib cells. This is done to preserve `connect_supply_net` (CSN) of secondary supplies to these dual-rail user lib cells; otherwise, the new CSN may be lost during comb tech mapping and multivoltage mapping.
- Prints the information message MV-055 to show the number of buffers/inverters converted from single-rail to dual-rail and vice versa.

  ```
  Information: Total 6 GTECH buffers have been converted from
   single-rail to dual-rail. (MV-055)
  Information: Total 1 GTECH buffer has been converted from dual-rail to
   single-rail. (MV-055)
  ```

  If no buffers/inverters of a specific single/dual-rail type are converted, MV-055 will not be printed for that type.

By default, the `mv.upf.ao_legalize_gtech_buf` option value is false and the AO legalization step is disabled.

Feedback

In create_mv_cells

To enable the AO legalization of GTECH buffers/inverters, at the same time as auto level-shifter insertion, set the application option `mv.upf.ao_legalize_gtech_buf` is set to true, before running the `create_mv_cells`.

```
set_app_options -as_user_default
   -name mv.upf.ao_legalize_gtech_buf -value true
```

When the is option enabled, during `create_mv_cells`, the tool performs the following steps:

- Runs the AO legalization at the same time as level-shifter insertion. This means greater flexibility to insert a new level shifter, convert a single-rail GTECH buffer/inverter into dual-rail, or do both. This flexibility will help to fix voltage violations which could not be solved previously because buffers/inverters act as hard terminal with fixed supplies. The flexibility allows to change supplies to these buffers/inverters by converting them into dual-rail.
- Converts single-rail buffers/inverters into dual-rail user lib cells, not dual-rail GTECH lib cells. This is done to preserve `connect_supply_net` (CSN) of secondary supplies to these dual-rail user lib cells; otherwise, the new CSN may be lost during comb tech mapping and multivoltage mapping.
- Prints the information message MV-055 to show the number of buffers/inverters converted from single-rail to dual-rail and vice versa.

  ```
  Information: Total 6 GTECH buffers have been converted from
   single-rail to dual-rail. (MV-055)
  Information: Total 1 GTECH buffer has been converted from dual-rail to
   single-rail. (MV-055)
  ```

  If no buffers/inverters of a specific single/dual-rail type are converted, MV-055 will not be printed for that type.

Extending AO Legalization to All Buffers and Inverters

To AO legalize both GTECH and non-GTECH buffers/inverters, use `create_mv_cells` with the `-always_on` option:

```
create_mv_cells -always_on
```

In this case, you need not set the application option `mv.upf.ao_legalize_gtech_buf` to true.

The `create_mv_cells -always_on` command runs only the AO legalization of buffers/inverters. No other multivoltage cells will be inserted.

Running `create_mv_cells -always_on -level_shifter` runs the AO legalization of buffers/inverters at the same time as auto level-shifter insertion. The will insert new level

shifters, convert single-rail buffers/inverters into dual-rail, or both, to fix voltage violations. This flexibility allows `create_mv_cells` to fix voltage violations that could not be solved previously.

By default, `create_mv_cells -always_on -level_shifter` gives a higher priority to minimizing the total number of level shifters over the total number of dual-rail buffers/inverters.

Moving Single-Rail Buffers to Nearby Voltage Areas

The `add_buffer_on_route` command inserts single-rail ECO buffers. However, the inserted ECO buffer has a multivoltage violation when the domain primary supply is different from the driver or load supply. In such cases, running the `fix_mv_design -buffer` command fixes the multivoltage violations involving the new buffer by converting the single-rail buffer to dual-rail.

However, when the user library does not have any dual-rail buffer library cells, the tool cannot perform the swapping, that is, the `fix_mv_design -buffer` command cannot convert the single-rail buffer into dual-rail. In some cases, no swapping is required and if the buffer is moved to a nearby voltage area, it can access the domain primary supply from that voltage area and remain as single-rail, resulting in a smaller area.

In such situations, to support the automatic movement of buffers to nearby voltage areas, for the buffers to access supply nets available in the other voltage areas, the `fix_mv_design -buffer` command provides two additional `-move_within_x` *`distance`* and `-move_within_y` *`distance`* options.

Here, *`distance`*:

- Is an integer multiple of the bounding box width (for `-move_within_x`) or height (`-move_within_y`) of the buffer
- Can be specified in both X- and Y-axis, or in only one of them

When you specify `-move_within_x` *`distance`*, `fix_mv_design -buffer` moves the buffer horizontally to the right or left within the specified distance. The actual horizontal distance moved is an integer multiple of the bounding box width of the buffer such that, after the buffer is moved, the entire buffer is in a different voltage area than that of the original location.

When you specify `-move_within_y` *`distance`*, `fix_mv_design -buffer` moves the buffer vertically upwards or downwards within the specified distance. The actual vertical distance moved is an integer multiple of the bounding box height of the buffer such that, after the buffer is moved, the entire buffer is in a different voltage area than that of the original location.

Feedback

Note:

The `-move_within_x` *`distance`* and `-move_within_y` *`distance`* options must be specified together with the `-buffer` option.

When using `fix_mv_design -buffer` with one or both of the move options, use the `-verbose` option to display the number of buffers or inverters that are moved, in the Buffer Tree Summary and the Buffer Tree Data tables. Also, with the `-verbose` option, the movement applied to each buffer or inverter is printed in the Comments column, in the detailed information of the fanout of the buffer tree that is fixed.

Limitations

The following limitations apply to moving single-rail buffers to nearby voltage areas using `-move_within_x` *`distance`* and `-move_within_y` *`distance`*:

- `fix_mv_design -move_within_x` *`distance`* and `-move_within_y` *`distance`* do not consider the following:
  - Whether or not the new location has enough space for the buffer
  - Whether or not the route to the new location is blocked or congested
  - The locations of the driver and fanouts of the buffer
- If a buffer is moved, you should run the `legalize_placement` command to finalize the location of the moved buffer and ensure legalization of your design.
- If a buffer is moved, the buffer then has a mismatch between the power domain and voltage area. This should be fixed by `fix_mv_design` successfully via logical feedthrough (LFT) or physical feedthrough (PFT). Therefore LFT or PFT must be enabled in the new voltage area with the `create_voltage_area_rule` command, for `-move_within_x` *`distance`* or `-move_within_y` *`distance`* to succeed. If not, the tool issues the MV-1096 warning message, for each voltage area without LFT and PFT enabled.
- If you specify a distance of 0, the tool uses the default distance value.
- If the `-update_power_domain` option is specified together with `-move_within_x` *`distance`* or `-move_within_y` *`distance`*, `-update_power_domain` always has a higher precedence.

Feedback

- If the driver supply is not available in the original power domain, `fix_mv_design` honors `-update_power_domain` and adds the buffer to the specified power domain. The tool then does not honor `-move_within_x` *`distance`* and `-move_within_y` *`distance`*.
- If the driver supply is available in the original power domain, `fix_mv_design` does not honor `-update_power_domain` but honors `-move_within_x` *`distance`* and `-move_within_y` *`distance`*.

UPF Support for Custom Always-On Wrapper Cells

The Fusion Compiler Design-For-Test (DFT) feature namely, custom wrapper cells on input signals, is used for the insertion of a test logic that wraps the existing input signal. Such a wrapper cell has a backup supply, to preserve the supply of the wrapped signal and thus its power characteristics. See the following figures:

Figure 70 Example RET_ctrl Always-on Control Signal

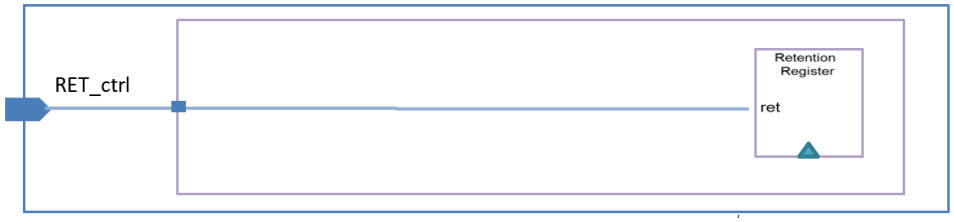
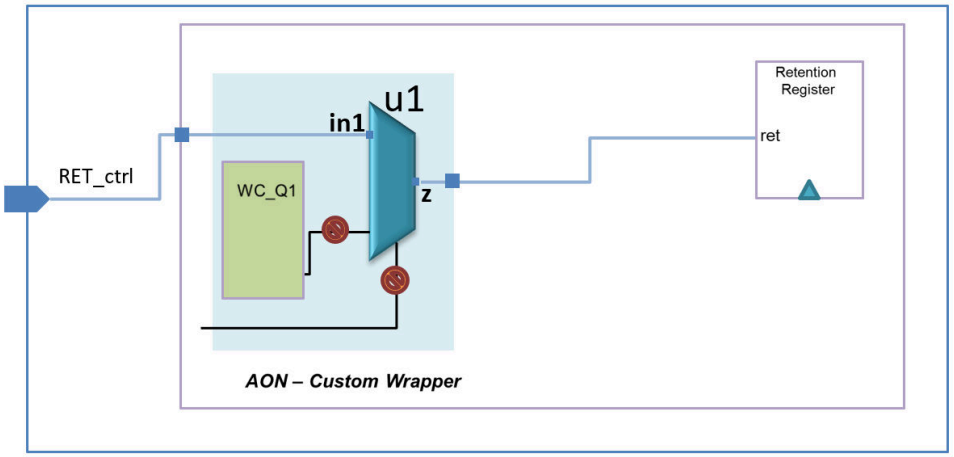


Figure 71 Example Wrapper Cell Wrapping RET_ctrl Control Signal, With Possible Isolation/LS Violations Marked



The Fusion Compiler UPF flow supports the insertion of custom always-on (AO) wrapper cells on input signals. The steps in the Fusion Compiler UPF flow for creating these custom AO multiplexer (MUX) cells are the following:

1. DFT calls the UPF before the wrapper cell insertion. The UPF tests the insertion and returns true or false depending on whether it allows the wrapper cell to be inserted in the given net segment
2. After the wrapper cell insertion, DFT calls the UPF again, to create the new UPF' statements for the newly-created wrapper cell. The Fusion Compiler UPF performs the following changes to the UPF for this new cell:
   - Add a `connect_supply_net` statement to the UPF to connect the backup power pin of the dual-rail MUX cell to the driver supply (or the local electrical equivalent) of the target net.
   - Add a `set_design_attribute` statement to the UPF to set the `leaf_cell_as_domain_boundary` attribute to `true` on the dual-rail MUX cell. This statement adds a leaf cell to the extent of the lower domain boundary of the parent domain of the cell. (See leaf_cell_as_domain_boundary Design Attribute.)
   - Add a `set_port_attribute` statement in the UPF to set the `upf_control_signal_trace` attribute to `true` on the appropriate input and output pins of the dual-rail MUX cell. This statement tells the UPF tools how to trace a control signal through the wrapper cell. (See upf_control_signal_trace Port Attribute.)

   These statements appear as tool-derived UPF statements.
3. The tool prepares to add the isolation and level-shifter cells (as necessary) on the dual-rail MUX's inputs, when the `insert_mv_cells` command is invoked.

leaf_cell_as_domain_boundary Design Attribute

The `leaf_cell_as_domain_boundary` design attribute can be set to `true` on leaf cell instances.

```
fc_shell> set_design_attributes -elements {cell_list}
             -attribute leaf_cell_as_domain_boundary TRUE
```

When a wrapper cell is inserted, this attribute is automatically inferred by the tool. When the value of this attribute is set to `true` for a cell instance, the tool considers this cell as a part of the lower boundary of its parent power domain. This means that the isolation and level shifter strategies apply to the cell.

This behavior is the same as the behavior specified in the LRM for macro cells, which is currently enabled in Synopsys implementation tools with the `macro_as_domain_boundary` attribute.

Feedback

Note:

If the `leaf_cell_as_domain_boundary` attribute is set to `true` on a cell that is not both always-on and a MUX leaf cell, the tool generates an error message. Do not set or use this attribute on any cell other than an always-on MUX leaf cell.

upf_control_signal_trace Port Attribute

The `upf_control_signal_trace` port attribute on signal ports, when set to `true`, ensures that the retention or isolation cells, driven by the signal which is wrapped by the always-on wrapper cell, continue to match their strategy after the cell is inserted.

```
fc_shell> set_port_attribute -ports {pin_list}
             -attribute upf_control_signal_trace TRUE
```

When tracing a UPF isolation or retention control signal for matching a strategy, the tool checks for this attribute on the driver pin of the control signal. If present, the tool treats the cell as a feedthrough path and continues tracing the control signal starting at the input pin with the corresponding attribute.

The value of the attribute is set to `true` on exactly one input pin and one output pin of an instance.

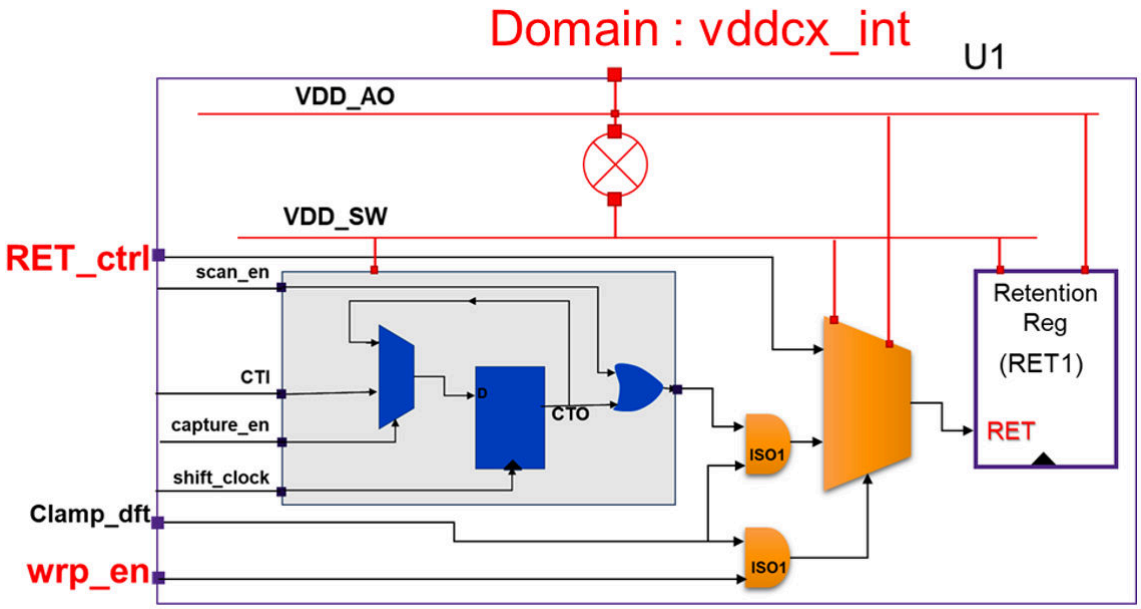
Note:

Do not set or use this attribute on any cell other than an always-on MUX leaf cell. The attribute can only be specified with the `-ports` option.

Feedback

Example

The following figure shows a design with an always-on wrapper cell added:



Before the insertion of this wrapper cell, the UPF tests whether to insert a wrapper signal on the RET_ctrl net, bringing in the wrp_en DFT control signal. The UPF determines that the VDD_AO supply is available in the vddcx_int domain and that the cell is bias-compatible with the domain bias supplies. The UPF then allows the DFT to proceed with the cell insertion.

After insertion of the cell, the tool infers the following statements and adds them to the UPF:

```
fc_shell> connect_supply_net VDD_AO -ports {U1/mux_inst/TVDD}
fc_shell> set_design_attributes -elements {U1/mux_inst}
            -attribute leaf_cell_as_domain_boundary TRUE
fc_shell> set_port_attributes -ports {U1/mux_inst/in1 U1/mux_inst/z}
            -attribute upf_control_signal_trace TRUE
```

During isolation insertion, the U1/mux_inst cell is recognized as part of the lower boundary of the vddcx_int domain, due to the `leaf_cell_as_domain_boundary` design attribute. Therefore, isolation cells are inserted according to the domain strategy.

When re-reading the design, if the tool needs to associate the retention register with a defined strategy, then the UPF traces the signal. When the signal trace reaches the U1/mux_inst/z pin, the `upf_control_signal_trace` port attribute is identified. Signal trace continues at the corresponding U1/mux_inst/in1 pin, finally arriving at RET_ctrl.

6

Well Biasing

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

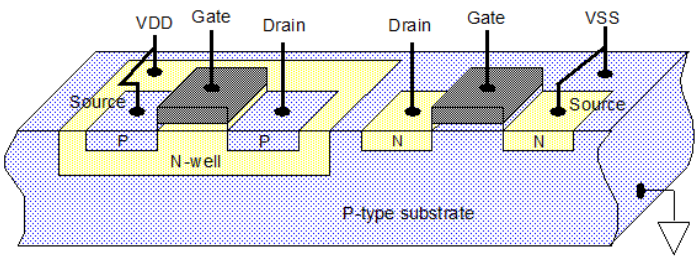
The synthesis and physical implementation tools offer an optional mode to specify the n-well and p-well bias supply infrastructure using UPF commands. In this mode, the tool automatically makes supply connections to the n-well and p-well bias pins.

This topic contains the following:

Purpose of Well Bias

In simpler CMOS process technologies, the NMOS transistors are typically built on a p-type silicon substrate and PMOS transistors are built in n-type wells embedded in the p-type substrate, as shown in Figure 72.

Figure 72 CMOS Technology Without Well Bias



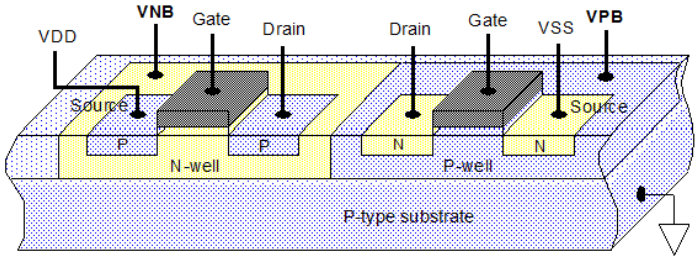
Feedback

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

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

Figure 73 shows how to build NMOS transistors in p-wells and PMOS transistors in n-wells, where the n-well and p-well voltages can be independently controlled. This type of construction requires the use of an advanced process technology.

Figure 73 CMOS Technology With N-Well and P-Well Bias



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

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

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

The well voltages can be modified dynamically to increase the speed when needed or to reduce leakage during standby mode. Alternatively, the bias voltage might be applied statically to compensate for process variations or for other adjustment purposes.

See Also

- Enabling Well Bias

Feedback

Library Modeling

In the Liberty-format description of a bias library cell, the UPF-based well bias mode allows only the following attribute definitions for bias supply pins:

- `pg_type`: `nwell` or `pwell`
- `direction`: `input`, `inout`, or `unspecified` (not `output` or `internal`)
- `physical_connection`: `routing_pin` or `device_layer`

Within each power domain, it is recommended that you consistently use library cells that have their `physical_connection` attribute set to either `routing_pin` or `device_layer`, not a mixture of both. When different physical connection parameters are defined in different libraries, you can use the `set_target_library_subset` command to restrict the selection of cells to specified libraries.

For more information about defining well bias PG pins, see the “PG Pin Syntax” section in the *Library Compiler User Guide*.

Library Cells With Only N-well Bias PG Pins

The tool supports library cells that only have n-well-biased PG pins. In this case, the tool behaves in the following way:

- Supply sets are created with p-well functions
- Standard power management cells with p-well PG pins are handled normally - power and ground pins of cells are implicitly connected to the respective supply set functions of primary, isolation, and retention supplies
- Cells with only n-well PG pins do no make the implicit connection to the p-well supply set functions
- Only the n-well connections are written out in the PG netlist
- Bias continuity and rail order checks are suppressed for p-well, if the p-well supply set function is not connected to PG pins
- The `check_mv_design` command warns you if the libraries have a mix of cells (some with the n-well PG pin only and other with both PG pin connections)

When using the `check_mv_design` command, you do not need to set the voltage on the p-well functions if it is not connected to any PG pins. If your libraries have standard cells with p-well PG pins, the tool requires that you set the voltage on p-well functions since these cells might be used during optimization.

Feedback

Enabling Well Bias

The well bias mode is disabled by default. To enable the UPF-based well bias mode, do the following:

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

You can selectively disable the well bias feature for lower-level blocks as follows:

```
fc_shell> set_design_attributes -elements {blkA blkB} -attribute
 enable_bias false
```

All supply sets created under the scope of a bias block include n-well and p-well functions. This is also true for implicit supply sets created for a power domain under the scope of a bias block.

Although you can *disable* the well bias feature for a block in a design where the feature is enabled, you cannot *enable* the well bias feature for a block in a design where the feature is disabled (either explicitly or by default). In other words, to enable this feature for a block, the feature must also be enabled for all parent blocks up to the top level.

Creating Bias Supplies

When you enable well biasing, the tool captures the `set_design_attributes -elements` command in the UPF' and UPF" files for consistent tool support across Synopsys tools. When you enable biasing and then create a supply set, the well bias supplies are automatically created as shown in Example 15.

Example 15 Creating Bias Supplies

```
set_design_attributes -elements {.} -attribute enable_bias TRUE
create_power_domain PD1 -include_scope
create_supply_set SS1
```

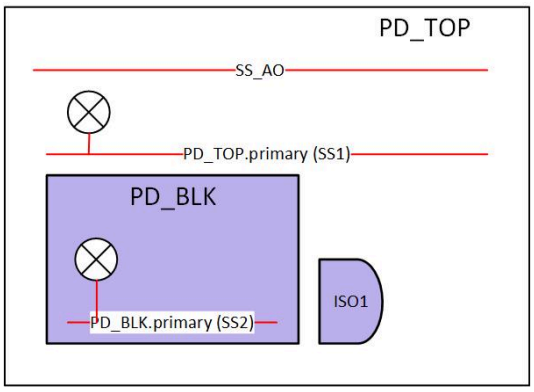
The commands in Example 15 create the following supplies:

- Power (SS1.power)
- Ground (SS1.ground)
- Pwell (SS1.pwell)
- Nwell (SS1.nwell)

Figure 74 illustrates how bias supplies are created when used with isolation cells.

Feedback

Figure 74 Bias Supplies and Isolation Cells



The UPF commands are as follows:

```
set_design_attributes -attribute {enable_bias} true
create_supply_set SS_AO
create_power_domain PD_TOP
creaet_power_domain PD_BLK
set_isolation -domain PD_BLK -isolation_supply SS_AO \
   -applies_to output -location parent
```

The commands create the following supplies:

- All four functions for PD_TOP (power, ground, nwell, and pwell)
- All four functions for PD_BLK (power, ground, nwell, and pwell)
- Two functions for SS_AO (power and ground)

The following connections are made for the isolation cell, ISO1:

Power	SS_AO.power
Ground	SS_AO.ground
Nwell	PD_TOP.primary.nwell
Pwell	PD_TOP.primary.pwell

Bias Design Rules

In the UPF-based well bias feature, a *bias library cell* is a logic library cell that has one or more bias pins defined as PG pins. A *nonbias library cell* is a logic library cell that does

Feedback

not define any bias pins. Bias and nonbias cells cannot be mixed within a given power domain. A hierarchical cell containing bias cells is called a *bias block*.

A nonbias block inside a bias design cannot use a bias supply set from a higher level of the design. It is restricted to using its own domain-dependent supply set, for example a domain created with the `{extra_supplies ""}create_power_domain -supply` command.

The tool performs the following checks:

- Rail Order Checks
  - The bias supply on a logic element must be more on than its related primary power supply
- Continuity Checks
  - All cells in a power domain must use the same bias supply (macros are an exception)
- Connectivity Checks
  - A power PG pin and a p-well PG pin cannot be connected to the same supply
  - A ground PG pin and an n-well PG pin cannot be connected to the same supply
  - An N-well PG pin and a p-well PG pin cannot be connected to the same supply
- Other Checks
  - Negative voltages are not allowed on n-well supply nets. They are permitted on p-well supply nets.

Bias Blocks

A bias block has the following characteristics:

- Any supply set (including an implicit supply set) created in the scope of the bias block has up to four supply functions: `power`, `ground`, `nwell`, and `pwell`. Some bias blocks use only the `power`, `ground`, and `nwell` functions.
- Some commands, such as `create_power_domain` and `create_supply_set`, treat the additional `nwell` and `pwell` supply functions like power and ground. Other commands, such as `set_domain_supply_net`, do not support the `nwell` and `pwell` functions and therefore cannot be used in the scope of the bias block.
- The target library must contain bias library cells. Nonbias library cells cannot be used in the bias block. However, nonbias macro cells are allowed, including pad cells.

Feedback

- During linking and optimization, the tool matches the voltages set on the supply functions (`power`, `ground`, `nwell`, and `pwell`) of the standard cells with the voltage map of the logic library cells.
- The connections to bias PG pins of standard cells are implicit. If you make an explicit connection to a bias PG pin of a bias cell using a supply other than the domain's default bias supply, you get a warning message with the `check_mv_design` command or a compile operation.
- During insertion of power management cells (isolation, level shifting, and retention), the tool considers the bias supplies in addition to power and ground.

The tool handles each nonbias block without considering any bias pins, even when the block exists in a bias design. During linking and optimization in a nonbias block, the tool ignores any bias pins in bias library cells.

In a nonbias design, the tool can use bias library cells because it only verifies that the `power` and `ground` supply functions of the cell match the design requirements; it ignores the bias supply pins of the cells. To ensure that only nonbias library cells are used in a nonbias design, use the `set_target_library_subset` command to restrict the selection of cells to nonbias libraries.

See Also

- Library Modeling
- Power Management Cell Insertion and Bias Supplies

7

Multivoltage Reporting and Debugging

The Fusion Compiler tool provides several commands that analyze and report multivoltage and power aspects of your design. See the following topics for more information:

- Checking the Design for Power Violations
- Reporting Multivoltage Paths
- Analyzing Level-Shifter Insertion
- Analyzing Unmapped Power Management Cells
- The Early Data Check Manager
- The Incomplete UPF Flow
- Reporting Commands for Multivoltage Designs
- Using the Verification Compiler Low Power Tool to Identify Problems

You can also use the Fusion Compiler Graphical User Interface multivoltage views for additional debugging. For more information, see the *Fusion Compiler Graphical User Interface User Guide*.

Checking the Design for Power Violations

The `check_mv_design` command performs a complete check of the power intent and PG connectivity. Use this command before the `compile_fusion` command to check power intent quality before synthesis and before you continue with physical synthesis. This command assesses the electrical correctness of your netlist. This command can also be run throughout the flow to ensure there are no violations as you synthesize the design.

You can restrict the types of rules to check with the `check_mv_design` command. For example, the `-isolation` option specifies to check isolation strategy and isolation cell rules, while the `-pg_pin` option specifies to check rules associated with PG pins.

Feedback

Reporting Multivoltage Paths

After you run the `compile_fusion` and `check_mv_design` commands, you can analyze multivoltage paths. The `report_mv_path` command reports detailed information about a multivoltage path through a specific object (pin, net or cell). For level-shifter and isolation cells, the command reports the reasons for insertion or association failures. For cells correctly inserted or associated, the command reports the supply, inputs, outputs, drivers, sinks, corresponding strategy, and corresponding power domain for each multivoltage cell.

For example, suppose the tool finds a missing isolation strategy as shown in Example 16.

Example 16 Missing Isolation Strategy

```
fc_shell> check_mv_design
************************************
Report : check_mv_design
Design : top
Version : ...
Date : ...
************************************
---------- Power domain rule ----------
No errors or warnings.

---------- Supply set rule -----------
No errors or warnings.

---------- Supply net rule -----------
No errors or warnings.

----------Isolation cell rule -----------
Warning: The isolation cell 'u0_0/pd_switch/DATA_UPF_ISO' is not
 associated with any isolation strategies. (MV-071)
Information: Total 1 MV-071 violations. (MV-080)

...
```

In this case, the `check_mv_design` command issues an error message for a power cell. To get a full report on the details of the specified cell, run the `report_mv_path` command with the `-cell` option:

```
fc_shell> report_mv_path -cell u0_0/pd_switch/DATA_UPF_ISO
```

Other options of the `report_mv_path` command provide a wide range of information. For example,

- The `-net` option reports the conditions on a path that includes the specified net.
- The `-pin` option reports the conditions on a path that includes the specified pin.

Feedback

- The `-all_heterogeneous_paths` option reports all heterogeneous paths in the design (paths where the related supplies are different among the load pins). When you use this option with the `report_mv_path` command, all other options are ignored.

  For each driver pin, the report includes all domain boundary ports and load pins with information about their related supplies. This information helps you to understand isolation cell insertion when the UPF has source- or sink-based isolation policies.

Analyzing Level-Shifter Insertion

To find out why level-shifter insertion failed, use the `analyze_mv_design -level_shifter` command. The command analyzes a whole path and reports all information regarding level-shifter insertion. The `-through` option accepts either a pin or net. If you specify this option, the tool analyzes the global net from that pin or net.

The tool displays the information in the following sections:

- Level-shifter application options - shows only those you have modified (nondefault) values
- Driver and load information - show the voltage range, power domain, related supplies, and the load fanout
- Path analysis

  Starting with the driver, the tool reports the path through each side of a power domain boundary.

For an overall report on level-shifter insertion failures, use the `-global_report` with the `-level_shifter` option. The tool reports the following errors:

- Wrong side of pin with the `-location self` option
- Error with the `-no_shift` strategy
- No library cell available

However, after the errors are collected, the tool selects only one error to report for each driver-load pair.

Analyzing Unmapped Power Management Cells

The `analyze_mv_feasibility` command analyzes whether the Fusion Compiler tool can map power management cells with the available logic libraries. If any cells cannot be mapped, the tool generates a report that provides details about every element that cannot be mapped.

The feasibility analysis is available for isolation cells (by using the `-isolation` option), enable level-shifter cells (by using the `-enable_level_shifter` option), and retention cells (by using the `-retention` option). In the absence of these options, the tool analyzes all three types of cells.

The recommended procedure is to use the `analyze_mv_feasibility` command after you read the RTL and load the UPF, but before you perform synthesis with the `compile_fusion` command. If you use the `analyze_mv_feasibility` command after power management cell insertion, the tool performs the analysis based on the current status of the netlist. The analysis results might be different in these two use cases because optimizations performed during the synthesis operation might prevent specific library cells from being used.

Complete sequential cell mapping (retention cells) is possible only after the compile operation. Before the compile operation, the `analyze_mv_feasibility` command reports whether any retention strategies do not define the control signals correctly. After the compile operation, the command generates a complete mapping report.

If any power management cells cannot be mapped, the tool issues a UPF-909 error message and returns a Tcl status of 0. In addition, the tool provides a text report about cells that cannot be mapped. The text report is similar to the following:

```
Iso/Els cell mapping failures:
--------------------------------------------------------------------------------
Library cell checking sequence:    Cell association with strategy
                                   Library cells set from strategy
                                   Dont_touch set by user or in library
                                   Site definition matching
                                   Purpose matching
                                   Process matching
                                   Voltage matching
                                   Temperature matching
                                   Bias matching
                                   Data inversion matching
                                   NOR inference rules
                                   Clamp value matching
                                   Sense value matching and phase
                                      correction checking for mismatch

Element(s) Reside  Strategy  Strategy  Clamp  Sense  Failure Reason
           Domain            Domain
--------------------------------------------------------------------------------
pin1       PDtop    iso1     PD1         1    high   Not included in strategy
 (boundary                                             mapped library
  port)                                                cells (114/115)
                                                     Data inverted mismatch
                                                       (1/1)
pin2       PD2      iso2     PD2         0    low    Not included in strategy
 (boundary                                             mapped library
  port)                                                cells (113/115)
                                                     PVT mismatch (2/2)
--------------------------------------------------------------------------------
```

An example of a retention cell text report is as follows:

```
Retention cell mapping failures:
--------------------------------------------------------------------------------
Retention cell mapping/library checking sequence:
                         Library cells set from map_retention_cell of strategy
                         Retention strategy control signal definition
                         Dont_touch set by user or in library
                         Zero pin retention library cell
                         Site definition matching
                         Purpose matching
                         Process matching
                         Voltage matching
                         Temperature matching
                         Bias matching
                         Data inversion matching
                         set_target_library_subset setting
                         Functionality matching
                         Clock phase matching
                         Scan equivalent checking
                         Complex retention instance dont_touch matching
                         Complex retention function id checking
                         Complex retention pins matching

Retention Element(s)     Reside  Strategy  Strategy   Failure Reason
                         Domain            Domain
--------------------------------------------------------------------------------
ret_cell_1 (cell)        PD1     RET1      PD1        Not included in strategy
                                                      mapped library
                                                      cells (45/69)
                                                    Purpose mismatch (23/23)
--------------------------------------------------------------------------------
```

The text report provides the number of library cells that were analyzed for mapping but does not report details about each of the individual library cells. To obtain more details about the mapping failures, use the `-format html` option to create detailed HTML reports.

HTML Cell Mapping Reports

The Fusion Compiler tool optionally provides detailed information about unmapped power management cells in HTML reports. HTML reporting is off by default. You can enable this feature as follows:

- To generate HTML reports at the `create_mv_cells -mapped` and `compile_fusion` commands, set the `mv.upf.enable_amvf_html_report` application option to `true` (the default is `false`).
- To generate HTML reports at the `analyze_mv_feasibility` command, use the `-format html` option. The tool honors this option regardless of the setting of the `mv.upf.enable_amvf_html_report` application option.

When HTML reporting is enabled and the design has unmapped power management cells, the tool creates a directory named pm_map.*x* in the run directory. The directory name *x* is an integer index that starts with 0 for the first report and increments by 1 for each

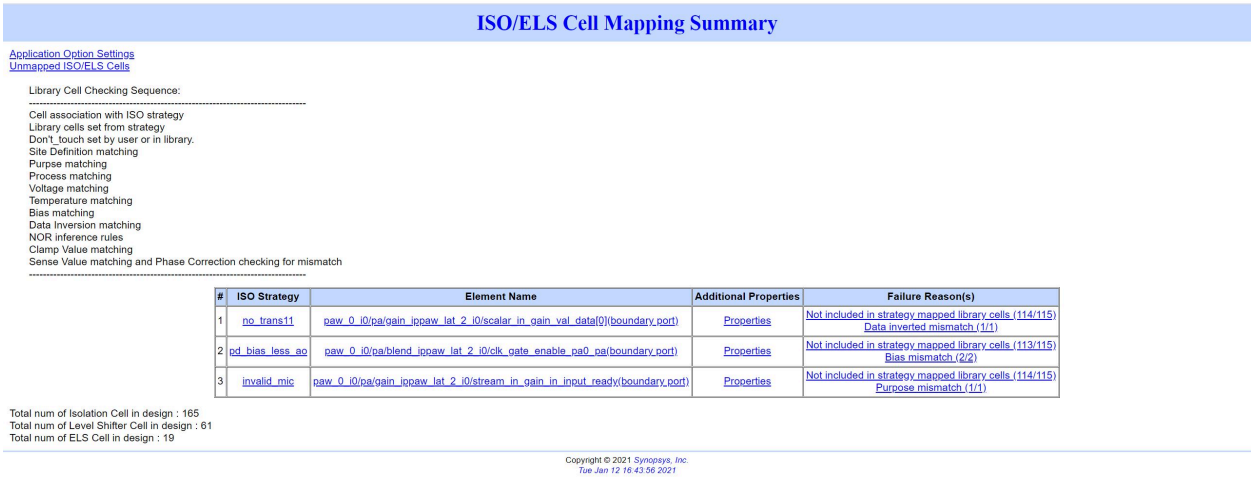
Feedback

additional report. Each execution of any of the listed commands generates a new report, which results in a new directory for that report.

In each pm_map.*x* directory, the tool writes an HTML file named pmMapper.html. The file is a top-level summary page with links to the details about unmapped cells.

Figure 75 is an example of the top-level summary page and Figure 76 is an example of the unmapped cell detail page. Many entries in the table are links to additional information.

Figure 75 Top-Level HTML Summary Page



ISO/ELS Cell Mapping Summary

Application Option Settings
Unmapped ISO/ELS Cells

Library Cell Checking Sequence:

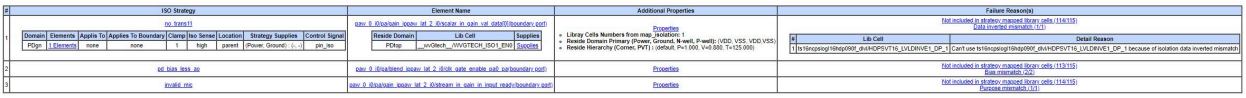
Cell association with ISO strategy
Library cells set from strategy
Don't_touch set by user or in library.
Site Definition matching
Purpse matching
Process matching
Voltage matching
Temperature matching
Bias matching
Data Inversion matching
NOR inference rules
Clamp Value matching
Sense Value matching and Phase Correction checking for mismatch
---

#	ISO Strategy	Element Name	Additional Properties	Failure Reason(s)
1	no_trans11	paw_0_i0/pa/gain_ippaw_lat_2_i0/scalar_in_gain_val_data[0](boundary port)	Properties	Not included in strategy mapped library cells (114/115) Data inverted mismatch (1/1)
2	pd_bias_less_ao	paw_0_i0/pa/blend_ippaw_lat_2_i0/clk_gate_enable_pa0_pa(boundary port)	Properties	Not included in strategy mapped library cells (113/115) Bias mismatch (2/2)
3	invalid_mic	paw_0_i0/pa/gain_ippaw_lat_2_i0/stream_in_gain_in_input_ready(boundary port)	Properties	Not included in strategy mapped library cells (114/115) Purpose mismatch (1/1)

Total num of Isolation Cell in design : 165
Total num of Level Shifter Cell in design : 61
Total num of ELS Cell in design : 19

Tue Jan 12 16:43:56 2021

Figure 76 Unmapped Cell Detail Page



The Early Data Check Manager

The Early Data Check Manager allows you to check designs for power and multivoltage issues early in the design cycle. You can configure different error conditions in different ways. The tool provides comprehensive reports on the data checks. The general flow is as follows:

1. Use the `set_early_data_check_policy` command to define the global violation handling policy for data checks.
2. Use the `set_early_data_check_policy` command to define the violation handling policy for specific data checks.
3. Proceed with your tool flow. The tool detects and responds to violations throughout the flow.

Feedback

4. Use the `report_early_data_checks` command to obtain a report about all violations or specific data checks.
5. Use the `get_early_data_check_records` command to get a Tcl collection of violations that can be used in other commands.
6. Use the `write_early_data_check_config` command to save the settings in a file.
7. Use the `remove_early_data_check_records` command to clear the data in preparation for another iteration.

For example, consider a UPF file named top.upf that contains the following statements:

```
associate_supply_set TOP.TOP_ISS_VDD90 -handle TOP.primary
add_power_state TOP.TOP_ISS_VDD90 -state active \
   {-supply_expr {power == '{FULL_ON, 0.72}} }
add_power_state TOP.primary -state active \
   {-supply_expr {power == '{FULL_ON, 0.9}} }
add_power_state -domain TOP -state S0 {-logic_expr \
   {primary==active&&PD_A.primary==active&&PD_B.primary&& \
   TOP.TOP_ISS_VDD90==active}}
```

In the Fusion Compiler session, execute the following commands to set all data check policies to their most lenient settings, read the UPF, and commit the UPF:

```
fc_shell> set_early_data_check_policy -policy lenient
...
fc_shell> load_upf top.upf
fc_shell> commit_upf
```

The action of committing the UPF causes the tool to evaluate some data checks. The tool finds a problem and issues the following statement:

```
Information: Policy for early data check 'mv.pst.conflict_supply_state'
 is 'tolerate'. (EDC-001)
```

Alternatively, you can set a more strict policy for this data check with the following command:

```
fc_shell> set_early_data_check_policy \
   -checks mv.pst.conflict_supply_state -policy error
```

As a result, the tool issues the following statement:

```
Information: Policy for early data check 'mv.pst.conflict_supply_state'
 is 'error'. (EDC-001)
Error: Conflicting supply states 'SNPS_INT_active' and
 'SNPS_INT_active_0' are specified for the
connected supply nets or ports 'TOP.primary.power' and
 'TOP.TOP_ISS_VDD90.power'. (UPF-159)
```

Feedback

For more information about the Early Data Check Manager, see the *Fusion Compiler Data Model User Guide*.

This section contains the following topics:

Multivoltage and Power Data Checks in the Early Data Check Manager

The `set_early_data_check_policy` command controls the actions of the Fusion Compiler tool when a violation is encountered. Individual data checks have up to three levels of allowable actions, which are specified by the policy settings `error`, `tolerate`, and `repair`. Global data check policy allows settings of `strict` and `lenient`.

To set a global policy, use the `-policy` option without the `-checks` option. In general, it is most efficient to first set a global policy, then modify individual data check policies as needed. The global policy settings are as follows:

- `strict`

  The tool sets all available data checks to the highest severity of policy, as supported by the individual data check.

- `lenient`

  The tool sets all available data checks to the lowest severity of policy, as supported by the individual data check.

To set a policy for one or more specific data checks, use the `-policy` option with the `-checks` option. Individual data check might not support all of these policy settings and the behavior of each data check might be different when a violation occurs. The policy settings are as follows, from most severe to least severe:

- `error`
- `tolerate`
- `repair`

After you set a policy, you cannot change the policy to a more strict setting unless you first execute the `reset_upf` command. The only exception is the mv.va.missing_voltage_area check.

The following tables describe the data checks, their supported policy settings, and the tool behavior when a violation is encountered. Table 23 describes data checks for supplies. Table 24 describes data checks for power strategies. Table 25 describes data checks

for power state tables. Table 26 describes data checks for hierarchical flows. Table 27 describes miscellaneous data checks. Table 28 describes how the tool infers supplies for power management cells. Table 29 describes the tool behavior for missing voltage areas.

Table 23 Data Checks for Supplies

Data Check	Policy	Tool Behavior
mv.supply.unknown_net_type	error	MV-042 warning (unable to derive power and ground type for a supply net)
	tolerate	Ignores the warning
mv.supply.inconsistent_resolution_type	error	UPF-030 error (inconsistent resolution type on a supply net)
	tolerate	Takes the first resolution type

Table 24 Data Checks for Power Strategies

Data Check	Policy	Tool Behavior
mv.strategy.invalid_lib_cell	error	UPF-108 or UPF-362 error (none of the library cells specified in a mapping or a strategy are legal)
	tolerate	Infers missing library cells from available library cells
mv.strategy.invalid_element	error	UPF-061 warning (none of the elements specified in a strategy are legal)
	tolerate	Ignores the strategy
mv.strategy.conflict_strategy	error	UPF-083 warning (a strategy conflicts with a power domain, such as when the same filters are applied twice)
	tolerate	Ignores the strategy
mv.strategy.unsupported_in_bias	error	UPF-209 error (an option is not supported in a bias block)
	tolerate	Ignores the data check

Feedback

Table 25 Data Checks for Power State Tables

Data Check	Policy	Tool Behavior
mv.pst.port_state_redefined	error	UPF-037 error (the same port state has been defined with different voltages)
	tolerate	Applies the first voltage to all equivalent supplies
mv.pst.conflict_power_state	error	UPF-056 error (the same power state has been defined with different voltages)
	tolerate	Applies the first voltage to all equivalent supplies
mv.pst.undefined_supply	error	UPF-067 error (a supply used in a power state table is undefined)
	tolerate	Ignores the supply
mv.pst.state_undefined_in_block	error	UPF-069 error (the top-level power state table includes a state that is not defined in the block power state table)
	tolerate	Ignores the power state or the entire block power state table
mv.pst.conflict_supply_state	error	UPF-159 error (conflicting supply states for equivalent supplies)
	tolerate	Takes the last state
mv.pst.power_state_undefined	error	UPF-171 error (a port or power state has not been defined)
	tolerate	Applies a wildcard

Table 26 Data Checks for Hierarchical Flows

Data Check	Policy	Tool Behavior
mv.hier.conflict_switch_strategies	error	UPF-144 (power domains from the top-level and block UPF cover the same element but cannot be merged due to conflicting UPF)
	tolerate	Ignores the strategy mismatch and merges the domains

Feedback

Table 27 Miscellaneous Data Checks

Data Check	Policy	Tool Behavior
mv.upf.invalid_terminal_boundary	error	UPF-170 error (a terminal boundary attribute is set on an object not on a domain boundary)
	tolerate	Ignores the attribute
mv.upf.invalid_bias_block	error	UPF-203 error (a bias block is defined under a non-bias block)
	tolerate	Ignores the error if the user-specified connection works
mv.buf.mv_violation	error	Inserts a buffer or inverter with the required supply nets
	tolerate	Allows buffering on clean nets without introducing new violations. Allows buffering on broken nets with either the driver or load supplies.
mv.cells.unmapped_gtech_cell	error	Keeps unmapped GTECH cells
	repair	Removes unmapped GTECH cells
opt.cells.incorrect_pvt	error	Sets the `opt.common.allow_incorrect_pvt` application option to `none`
	tolerate	Sets the `opt.common.allow_incorrect_pvt` application option to `pm_cells`

Table 28 Inferring Supplies for Power Management Cells

Cell Type	Data Check	Policy	Strategies
Isolation	mv.cells.iso_missing_csn	repair	infer_from_primary (default) infer_from_sink infer_from_control_signal
Level shifter	mv.cells.ls_missing_csn	repair	infer_from_primary (default) infer_input_from_source_output_from_sink
Retention	mv.cells.ret_missing_csn	repair	infer_from_primary (default) infer_from_most_always_on_supply

Feedback

Table 28 Inferring Supplies for Power Management Cells (Continued)

Cell Type	Data Check	Policy	Strategies
Power switch	mv.cells.psw_missing_csn	repair	infer_from_primary (default) infer_from_most_always_on_supply
Always-on	mv.cells.ao_missing_csn	repair	infer_from_primary (default) infer_from_driver_or_load
Other non-primary PG pin	mv.cells.other_missing_csn	repair	infer_from_primary (default)

Table 29 Data Checks for Missing Voltage Areas

Data Check	Policy	Strategy	Tool Behavior
mv.va.missing_voltage_area	error	n/a	MV-511 or MV-511a error
	repair	infer_va_for_synthesis_only	Enables missing voltage area for non-action and synthesis commands
		infer_va	Enables missing voltage area for all commands

Reporting Data Checks

Use the `report_early_data_checks` command to list the failing checks. The default report lists the data check name, the policy in effect, and the number of violations. The verbose report provides the text of warning or error messages for each violation along with a brief description of any action taken by the tool. The reports include only one record per checked design object.

An example of a default report is as follows:

```
Check                                Policy        Strategy     Fail Count
-------------------------------------------------------------------------
mv.pst.conflict_supply_state         tolerate
mv.va.missing_voltage_area           repair        infer_va       1
-------------------------------------------------------------------------
```

An example of a verbose report is as follows:

```
Check                          Policy    Strategy    Checked     Comment
                                                     Objects
-------------------------------------------------------------------------
mv.pst.conflict_supply_state  tolerate             TOP.power   Error:
Conflicting supply states 'active' and 'active_0' are specified for the
connected supply nets or ports 'TOP.primary.power' and 'TOP.ISS.power'.
(UPF-159j). Take the later state

mv.va.missing_voltage_area    repair    infer_va    PD_A        The power
domain is associated to the voltage area DEFAULT_VA
-------------------------------------------------------------------------
```

Use the `get_early_data_check_records` command to create a collection of data check violations. The results can be used in other Tcl commands. The format is a space-delimited list of checks with the format check@object. For example:

```
fc_shell> get_early_data_check_records
{mv.pst.conflict_supply_state@TOP.TOP_ISS_VDD90
 mv.va.missing_voltage_area@PD_A}
```

You can use the `-filter` option to filter the collection and the `-hierarchical` option to traverse the entire design.

The Incomplete UPF Flow

The Data Check Manager provides a unified method for checking and responding to power and multivoltage violations when the design is incomplete.

The legacy incomplete UPF flow described in this section is disabled if you use any of the data check manager commands. For more information, see The Early Data Check Manager.

The incomplete UPF flow has the following minimum requirements:

- Power domains must be defined according to current scoping requirements.
- Primary supplies must be defined in all domains.
- PVT information must be specified.

You can have incomplete power state tables and incomplete level-shifter, isolation, retention, and power switch strategies.

Generally, there are two ways the tool handles incomplete UPF:

1. The tool ignores errors and continues with the flow
2. The tool derives the necessary UPF intent for the flow to continue

Feedback

To enable the incomplete UPF flow, set the `mv.incomplete_upf.enable` application option to `true` (the default is `false`).

Topics covered in this section:

- Ignoring UPF Errors
- Deriving Missing Supplies
- Deriving Isolation and Level-Shifter Library Cells
- Missing Voltage Areas

Ignoring UPF Errors

Table 30 summarizes the errors that the tool ignores and the action taken to allow the incomplete UPF flow to continue.

Table 30 List of Ignored UPF Errors

UPF Error	Action Taken
Unable to derive power and ground type for supply net	Ignores this only if the supply net is not used in the current design
Inconsistent resolution type on supply net	Uses the first resolution type
Port state already defined	Uses the first port state defined and applies the voltage to all equivalent supplies
Conflicting power states	Uses the first power state defined and applies the voltage to all equivalent supplies
No valid elements specified in the strategy	Ignores this strategy
Undefined supply used in power state table	Ignores this supply
Power state not fully included in a lower-level power state table	Ignores the conflicting states or the entire block power state table
Strategy conflicts with another strategy	Ignores this strategy
Power cell is not associated with any power strategies or supply net connections	Derives the supply net for this cell based on other UPF constraints
Domains cannot be merged	Ignores the power switch strategy mismatch
Conflicting supply states	Uses the latter supply state
Terminal boundary attribute set on a boundary that is not a domain boundary	Ignores the attribute

Feedback

Table 30 List of Ignored UPF Errors (Continued)

UPF Error	Action Taken
Undefined port or power state in the power state table	Applies wildcard
Bias block defined under a nonbiased block	Ignores bias block definition
Unsupported option for a bias block	Ignores check

Deriving Missing Supplies

In incomplete UPF mode, the tool infers supply net connections for isolation, level-shifter, retention, power switch, and always-on cells. The tool also infers the power supply for unspecified backup PG pins. The tool uses the primary supply unless you specify otherwise.

Table 31 lists the application options you can set for these rules.

Table 31 Inferring Supplies for Power Management Cells

Cell Type	Application Option	Applicable Rules
Isolation	mv.incomplete_upf.iso_infer_csn_rule	infer_from_control_signal infer_from_primary (default) infer_from_sink
Level shifter	mv.incomplete_upf.ls_infer_csn_rule	infer_input_from_source_output_from_sink infer_from_primary (default)
Retention	mv.incomplete_upf.ret_infer_csn_rule	infer_from_most_always_on_supply infer_from_primary (default)
Power switch	mv.incomplete_upf.psw_infer_csn_rule	infer_from_most_always_on_supply infer_from_primary (default)
Always-on	mv.incomplete_upf.ao_infer_csn_rule	infer_from_driver_or_load infer_from_primary (default)

Deriving Isolation and Level-Shifter Library Cells

When you enable the incomplete UPF mode and no library cell is valid in a `map_isolation_cell` or `map_level_shifter_cell` command, the tool tries to infer the library cell from available isolation and level-shifter library cells. The `save_upf` command keeps the original input commands.

The `report_incomplete_upf` command reports any missing or invalid library cells.

Missing Voltage Areas

By default, the Fusion Compiler tool allows missing voltage areas whether or not you are using the incomplete UPF flow. In the Fusion Compiler tool, all reporting, checking, querying, and UPF commands do not need the voltage areas to be defined. However, the `check_mv_design` and `check_design` commands still report missing voltage area warnings when this feature is enabled.

If power domains are defined at the top level but not propagated from a lower level, the tool assigns DEFAULT_VA to the top-level power domains that do not have a user-defined voltage area.

For power domains inside a lower-level block, the tool behavior is described in Table 32 with respect to the setting of the `mv.upf.enable_missing_voltage_area` application option for the blocks (the default is `true`).

To disable missing voltage area support, set the `mv.upf.enable_missing_voltage_area` application option to `false`.

Feedback

Table 32 Handling of Missing Voltage Areas

Top-Level Block	Lower-Level Block	Behavior for Power Domain in Lower-Level Block With a Missing Voltage Area
Enabled	Enabled	Assigned to DEFAULT_VA of the physical block. At the top level, the corresponding power domain is assigned to the voltage area that represents the DEFAULT_VA of the lower-level block.
Disabled	Disabled	Not assigned to DEFAULT_VA in either the top-level or lower-level block. Implementation commands that require voltage areas result in an MV-51 error in both blocks.
Enabled	Disabled	Not assigned to DEFAULT_VA in either the top-level or lower-level block. Implementation commands that require voltage areas result in an MV-51 error in both blocks.
Disabled	Enabled	Assigned to DEFAULT_VA of the physical block. At the top level, the corresponding power domain is assigned to the voltage area that represents the DEFAULT_VA of the lower-level block.

Reporting Commands for Multivoltage Designs

The following commands are helpful for looking up UPF-related information:

- The `report_cells -power` command displays the PG pin connectivity for a given instance.
- The `report_power_domains` command displays the power intent information for the specified domains or all power domains.

  The command also reports if the specified domain is an atomic power domain.
- The `report_mv_lib_cells` command displays multivoltage library cell information. You can specify the format for your report (HTML or CSV).
- The `report_mv_cells` command displays information related to multivoltage cells in the design.
- The `report_mv_design` command displays a summary of the design including the number of power domains, number of supply nets, number of supply sets, details of the voltage areas, and the number and type of power management cells.
- The `report_mv_path` command reports paths with multivoltage constraints and the associated multivoltage cells.
- The `report_lib_pins` command displays pin information for a specified library. You can set up the report configuration using the `set_report_configuration` command.

- The `report_pst` command displays the power state table for the design.
- The `report_supply_ports` command reports detailed information about supply port connectivity.
- The `report_supply_nets` command reports detailed information about a supply net, such as its states and connected ports.
- The `report_supply_sets` command reports detailed information about the specified supply sets in the current scope.
- The `check_equivalent_power_domains` command checks whether specified power domains are equivalent.
- The `get_equivalent_power_domains` command displays a list of equivalent power domains.

Using the Verification Compiler Low Power Tool to Identify Problems

The Synopsys Verification Compiler Low Power tool (VC LP tool) is a static low power rule checker. It performs syntax and semantic checks on the UPF as well as structure checks on the insertion and connection of power management cells.

The Fusion Compiler tool can use the VC LP tool to identify UPF and related violations early in the design cycle. If there are no violations, you can proceed with implementation.

If violations exist, you can use ECO commands to fix them. However, only a limited subset of ECO commands is supported for design fixing. You cannot add or change UPF strategies or power domains to fix violations. This method does not support large netlist changes.

Note:

This method supports full flat verification in the VC LP tool. To perform top-only verification, you must manually edit the template Tcl file.

The general procedure is as follows:

1. Set the `VC_STATIC_HOME` environment variable to the location of the Verification Compiler tool.
2. (Optional) If you want to change any default settings, create a template Tcl file and specify it as follows:

   ```
   fc_shell> setenv ICC2_VCLP_TEMPLATE $template_file
   ```

3. Open a design block.

4. (Optional) Configure the VC LP tool environment by using the `set_vclp_options` command to change default settings. Examine the settings by using the `report_vclp_options` command.
5. Use the `check_vclp_design` command to analyze the block.
6. Examine the VC LP report file.
7. If no violations are found, proceed with implementation.
8. If violations exist, debug them by checking the Fusion Compiler layout view or the Verification Compiler schematic view. You can also execute a Verification Compiler command by using the `run_vclp_cmd` command in the Fusion Compiler tool.
9. Use the `vclp_zoom_highlight` command to highlight a selected instance.
10. Fix violations by using supported ECO commands and check the design again.
11. (Optional) Exit the Verification Compiler environment by using the `vclp_stop` command. Otherwise, the shell automatically stops when you exit the Fusion Compiler shell.

When you execute the `check_vclp_design` command, the Fusion Compiler tool writes the UPF and Verilog files and submits them as input to the VC LP tool.

If you are using the golden UPF flow, the VC LP tool must also run in the golden UPF mode. In this case, use the `-golden_upf_files` option with the `check_vclp_design` command. The argument for this option is a list of golden UPF file paths (full paths) enclosed in double quotation marks.

You can optionally use the `-write_verilog_options` option with the `check_vclp_design` command to specify any of the options available with the `write_verilog` command. Similarly, you can use the `-save_upf_options` option to specify any of the options available with the `save_upf` command. You must use both of these options together. If specified, these options take precedence over the default behavior.

If the design has any hierarchical instances, such as extracted timing models or budget shells, the `-write_verilog_options` and `-save_upf_options` options are required. The following commands are examples for hierarchical designs:

```
fc_shell> check_vclp_design -write_verilog_options \
   "-hierarchy all -exclude {all_physical_cells}" -save_upf_options \
   "-exclude {all_physical_cells}"

fc_shell> check_vclp_design -write_verilog_options \
   "-hierarchy all -switch_view_list design" -save_upf_options \
   "-full_chip"
```

Feedback

The default behavior of the `-write_verilog_options` option is equivalent to the following command:

```
write_verilog -exclude {leaf_module_declarations corner_cells \
   cover_cells end_cap_cells filler_cells pad_spacer_cells \
   physical_only_cells} -hierarchy all
```

The default behavior of the `-save_upf_options` option is equivalent to the following command:

```
save_upf -exclude {corner_cells cover_cells end_cap_cells filler_cells \
   pad_spacer_cells physical_only_cells}
```

The following ECO commands are supported for netlist and UPF changes: `create_port`, `remove_ports`, `create_net`, `remove_nets`, `create_port_bus`, `remove_port_buses`, `create_net_bus`, `remove_net_buses`, `connect_net`, `disconnect_net`, `remove_buffers`, `create_cell`, `remove_cells`, `change_link`, and `connect_supply_net`.