IEEE Standard for Design and Verification of Low-Power Integrated Circuits

IEEE Computer Society

Sponsored by the Design Automation Committee and the IEEE Standards Association Corporate Advisory Group

IEEE 3 Park Avenue New York, NY 10016-5997 USA

IEEE Std 1801[™]-2013 (Revision of IEEE Std 1801-2009)

29 May 2013

IEEE Standard for Design and Verification of Low-Power Integrated Circuits

Sponsor

Design Automation Committee of the IEEE Computer Society and the IEEE Standards Association Corporate Advisory Group

Approved 6 March 2013 IEEE-SA Standards Board Grateful acknowledgment is made to the following for permission to use source material:

Accellera Systems Initiative Unified Power Format (UPF) Standard, Version 1.0

Cadence Design Systems, Inc.

Library Cell Modeling Guide Using CPF Hierarchical Power Intent Modeling Guide Using CPF

Silicon Integration Initiative, Inc.

Si2 Common Power Format Specification, Version 2.0

Abstract: A method is provided for specifying power intent for an electronic design, for use in verification of the structure and behavior of the design in the context of a given power management architecture, and for driving implementation of that power management architecture. The method supports incremental refinement of power intent specifications required for IP-based design flows. Keywords: corruption semantics, IEEE 1801™, interface specification, IP reuse, isolation, levelshifting, power-aware design, power domains, power intent, power modes, power states, progressive design refinement, retention, retention strategies

Verilog is a registered trademark of Cadence Design Systems, Inc.

Print: ISBN 978-0-7381-8282-7 STDPD98167 PDF: ISBN 978-0-7381-8281-0 STDGT98167

The Institute of Electrical and Electronics Engineers, Inc. 3 Park Avenue, New York, NY 10016-5997, USA

Copyright © 2013 by The Institute of Electrical and Electronics Engineers, Inc. All rights reserved. Published 29 May 2013. Printed in the United States of America.

IEEE is a registered trademark in the U.S. Patent & Trademark Office, owned by The Institute of Electrical and Electronics Engineers, Incorporated.

IEEE prohibits discrimination, harassment and bullying.

For more information, risit <u>http://www.ieee.org/web/aboutus/whatis/policies/p9-26.html</u>. No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

Notice and Disclaimer of Liability Concerning the Use of IEEE Documents: IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While IEEE administers the process and establishes rules to promote fairness in the consensus development process, IEEE does not independently evaluate, test, or verify the accuracy of any of the information or the soundness of any judgments contained in its standards.

Use of an IEEE Standard is wholly voluntary. IEEE disclaims liability for any personal injury, property or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon any IEEE Standard document.

IEEE does not warrant or represent the accuracy or content of the material contained in its standards, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained in its standards is free from patent infringement. IEEE Standards documents are supplied "AS IS."

The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE standard is subjected to review at least every ten years. When a document is more than ten years old and has not undergone a revision process, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE standard.

In publishing and making its standards available, IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing any IEEE Standards document, should rely upon his or her own independent judgment in the exercise of reasonable care in any given circumstances or, as appropriate, seek the advice of a competent professional in determining the appropriateness of a given IEEE standard.

Translations: The IEEE consensus development process involves the review of documents in English only. In the event that an IEEE standard is translated, only the English version published by IEEE should be considered the approved IEEE standard.

Official Statements: A statement, written or oral, that is not processed in accordance with the IEEE-SA Standards Board Operations Manual shall not be considered the official position of IEEE or any of its committees and shall not be considered to be, nor be relied upon as, a formal position of IEEE. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that his or her views should be considered the personal views of that individual rather than the formal position of IEEE.

Comments on Standards: Comments for revision of IEEE Standards documents are welcome from any interested party, regardless of membership affiliation with IEEE. However, IEEE does not provide consulting information or advice pertaining to IEEE Standards documents. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Since IEEE standards represent a consensus of concerned interests, it is important to ensure that any responses to comments and questions also receive the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to comments or questions except in those cases where the matter has previously been addressed. Any person who would like to participate in evaluating comments or revisions to an IEEE standard is welcome to join the relevant IEEE working group at <u>http://standards.ieee.org/develop/wg/</u>.

Comments on standards should be submitted to the following address:

Secretary, IEEE-SA Standards Board 445 Hoes Lane Piscataway, NJ 08854 USA

Photocopies: Authorization to photocopy portions of any individual standard for internal or personal use is granted by The Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Notice to users

Laws and regulations

Users of IEEE Standards documents should consult all applicable laws and regulations. Compliance with the provisions of any IEEE Standards document does not imply compliance to any applicable regulatory requirements. Implementers of the standard are responsible for observing or referring to the applicable regulatory regulatory requirements. IEEE does not, by the publication of its standards, intend to urge action that is not in compliance with applicable laws, and these documents may not be construed as doing so.

Copyrights

This document is copyrighted by the IEEE. It is made available for a wide variety of both public and private uses. These include both use, by reference, in laws and regulations, and use in private self-regulation, standardization, and the promotion of engineering practices and methods. By making this document available for use and adoption by public authorities and private users, the IEEE does not waive any rights in copyright to this document.

Updating of IEEE documents

Users of IEEE Standards documents should be aware that these documents may be superseded at any time by the issuance of new editions or may be amended from time to time through the issuance of amendments, corrigenda, or errata. An official IEEE document at any point in time consists of the current edition of the document together with any amendments, corrigenda, or errata then in effect. In order to determine whether a given document is the current edition and whether it has been amended through the issuance of amendments, corrigenda, or errata, visit the IEEE-SA Website at http://standards.ieee.org/index.html or contact the IEEE at the address listed previously. For more information about the IEEE Standards Association or the IEEE standards development process, visit IEEE-SA Website at http://standards.ieee.org/index.html or *index.html*.

Errata

Errata, if any, for this and all other standards can be accessed at the following URL: <u>http://standards.ieee.org/findstds/errata/index.html</u>. Users are encouraged to check this URL for errata periodically.

Patents

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken by the IEEE with respect to the existence or validity of any patent rights in connection therewith. If a patent holder or patent applicant has filed a statement of assurance via an Accepted Letter of Assurance, then the statement is listed on the IEEE-SA Website at http://standards.ieee.org/about/sasb/patcom/patents.html. Letters of Assurance may indicate whether the Submitter is willing or unwilling to grant licenses under patent rights without compensation or under reasonable rates, with reasonable terms and conditions that are demonstrably free of any unfair discrimination to applicants desiring to obtain such licenses.

Essential Patent Claims may exist for which a Letter of Assurance has not been received. The IEEE is not responsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity or scope of Patents Claims, or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance, if any, or in any licensing agreements are reasonable or non-discriminatory. Users of this standard are expressly advised that determination of the validity of any patent rights, and the risk of infringement of such rights, is entirely their own responsibility. Further information may be obtained from the IEEE Standards Association.

Participants

The Unified Power Format Working Group is entity based. At the time this standard was completed, the Unified Power Format Working Group had the following membership:

John Biggs, Chair Erich Marschner, Vice Chair Jeffrey Lee, Secretary Joe Daniels, Technical Editor

Dave Allen Ido Bourstein Shir-Shen Chang David Cheng Cary Chin Sumit DasGupta Sorin Dobre Shaun Durnan Colin Holehouse Sushma Honnavara-Prasad Fred Jen Tim Jordan Knut Just James Kehoe Rick Koster Rolf Lagerquist Lisa McIlwain Don Mills Barry Pangrle Albert Rich Judith Richardson Jim Sproch Amit Srivastava Prasad Subbarao Venki Venkatesh Qi Wang Jon Worthington Louis Yu

The following members of the entity balloting committee voted on this standard. Balloters may have voted for approval, disapproval, or abstention.

- Accellera Systems Initiative Advanced Micro Devices (AMD) ARM, Ltd. Atrenta Inc. Broadcom Corporation Cadence Design Systems, Inc. Cambridge Silicon Radio Cortina Systems Intel Corporation Japan Electronics and Information Technology Industries Association (JEITA)
- LSI Corporation Marvell Technology Group Ltd. MediaTek Inc. Mentor Graphics Corporation Qualcomm Incorporated Silicon Integration Initiative, Inc. STMicroelectronics Synopsys, Inc. Texas Instruments Incorporated Xilinx

When the IEEE-SA Standards Board approved this standard on 6 March 2013, it had the following membership:

John Kulick, Chair David J. Law, Vice Chair Richard H. Hulett, Past Chair Konstantinos Karachalios, Secretary

Masayuki Ariyoshi Peter Balma Farooq Bari Ted Burse Wael William Diab Stephen Dukes Jean-Philippe Faure Alexander Gelman Mark Halpin Gary Hoffman Paul Houzé Jim Hughes Michael Janezic Joseph L. Koepfinger* Oleg Logvinov Ron Petersen Gary Robinson Jon Walter Rosdahl Adrian Stephens Peter Sutherland Yatin Trivedi Phil Winston Yu Yuan

*Member Emeritus

Also included are the following nonvoting IEEE-SA Standards Board liaisons:

Richard DeBlasio, *DOE Representative* Michael Janezic, *NIST Representative*

Julie Alessi IEEE Standards Program Manager, Document Development

Krista Gluchoski IEEE Standards Program Manager, Technical Program Development

Introduction

This introduction is not part of IEEE Std 1801-2013, IEEE Standard for Design and Verification of Low-Power Integrated Circuits.

The purpose of this standard is to provide portable low-power design specifications that can be used with a variety of commercial products throughout an electronic system design, analysis, verification, and implementation flow.

When the electronic design automation (EDA) industry began creating standards for use in specifying, simulating, and implementing functional specifications of digital electronic circuits in the 1980s, the primary design constraint was the transistor area necessary to implement the required functionality in the prevailing process technology at that time. Power considerations were simple and easily assumed for the design as power consumption was not a major consideration and most chips operated on a single voltage for all functionality. Therefore, hardware description languages (HDLs) such as VHDL (IEC 61691-1-1/ IEEE Std 1076^{TM})^a and SystemVerilog (IEEE Std 1800^{TM}) provided a rich set of capabilities necessary for capturing the functional specification of electronic systems, but no capabilities for capturing the power architecture (how each element of the system is to be powered).

As the process technology for manufacturing electronic circuits continued to advance, power (as a design constraint) continually increased in importance. Even above the 90 nm process node size, dynamic power consumption became an important design constraint as the functional size of designs increased power consumption at the same time battery-operated mobile systems, such as cell phones and laptop computers, became a significant driver of the electronics industry. Techniques for reducing dynamic power consumption—the amount of power consumed to transition a node from a 0 to 1 state or vice versa—became commonplace. Although these techniques affected the design methodology, the changes were relatively easy to accommodate within the existing HDL-based design flow, as these techniques were primarily focused on managing the clocking for the design (more clock domains operating at different frequencies and gating of clocks when logic in a clock domain is not needed for the active operational mode). Multi-voltage power-management methods were also developed. These methods did not directly impact the functionality of the design, requiring only level-shifters between different voltage domains. Multi-voltage power domains could be verified in existing design flows with additional, straight-forward extensions to the methodology.

With process technologies below 100 nm, static power consumption has become a prominent and, in many cases, dominant design constraint. Due to the physics of the smaller process nodes, power is leaked from transistors even when the circuitry is quiescent (no toggling of nodes from 0 to 1 or vice versa). New design techniques were developed to manage static power consumption. Power gating or power shut-off turns off power for a set of logic elements. Back-bias techniques are used to raise the voltage threshold at which a transistor can change its state. While back bias slows the performance of the transistor, it greatly reduces leakage. These techniques are often combined with multi-voltages and require additional functionality: power-management controllers, isolation cells that logically and/or electrically isolate a shutdown power domain from "powered-up" domains, level-shifters that translate signal voltages from one domain to another, and retention registers to facilitate fast transition from a power-off state to a power-on state for a domain.

The EDA industry responded with multiple vendors developing proprietary low-power specification capabilities for different tools in the design and implementation flow. Although this solved the problem locally for a given tool, it was not a global solution in that the same information was often required to be specified multiple times for different tools without portability of the power specification. At the Design

^aInformation on references can be found in Clause 2.

Automation Conference (DAC) in June 2006, several semiconductor/electronics companies challenged the EDA industry to define an open, portable power specification standard. The EDA industry standards incubation consortium, Accellera Systems Initiative, answered the call by creating a Technical SubCommittee (TSC) to develop a standard. The effort was named Unified Power Format (UPF) to recognize the need of unifying the capabilities of multiple proprietary formats into a single industry standard. Accellera approved *UPF 1.0* as an Accellera standard in February 2007. In May 2007, Accellera donated UPF to the IEEE for the purposes of creating an IEEE standard, and in March 2009, the first version of the IEEE Std 1801 was released. So this standard, although the second version of the IEEE Std 1801, represents the third version of what is more colloquially referred to as UPF.

Contents

1.	Over	Overview			
	1.1	Scope	1		
	1.2	Purpose	1		
	1.3	Key characteristics of the Unified Power Format	1		
	1.4	Use of color in this standard	3		
	1.5	Contents of this standard	3		
	1.0				
2.	Norn	native references	4		
3.	Defir	itions, acronyms, and abbreviations	4		
	3.1	Definitions	4		
	3.2	Acronyms and abbreviations	9		
4.	UPF concepts				
	4.1	Design structure	. 11		
	4.2	Design representation	. 11		
	4.3	Power architecture	. 14		
	4.4	Power distribution	. 17		
	4.5	Power management	. 23		
	4.6	Power states	. 26		
	4.7	Simstates	. 29		
	4.8	Successive refinement	. 30		
	4.9	Tool flow	. 31		
	4.10	File structure	. 32		
5.	Language basics				
	5.1	UPF is Tcl	. 33		
	5.2	Conventions used	. 33		
	5.3	Lexical elements	. 34		
	5.4	Boolean expressions	. 37		
	5.5	Object declaration	. 39		
	5.6	Attributes of objects	. 40		
	5.7	Power state name spaces	. 43		
	5.8	Precedence	. 44		
	5.9	Generic UPF command semantics	. 45		
	5.10	effective element list semantics	. 45		
	5.11	Command refinement	. 48		
	5.12	Error handling	. 49		
	5.13	Units	. 50		
6.	Power intent commands				
	6.1	Categories	. 51		
	6.2	add_domain_elements [deprecated]	51		
	6.3	add_port_state [legacy]	. 52		
	6.4	add_power_state	. 52		
	6.5	add_pst_state [legacy]	. 57		
	6.6	apply_power_model	. 58		
	6.7	associate_supply_set	. 59		

6.8	begin_power_model	60			
6.9	bind_checker	61			
6.10	connect_logic_net	63			
6.11	connect_supply_net	64			
6.12	connect_supply_set	65			
6.13	create composite domain	67			
6.14	create hdl2upf vct	68			
6.15	create logic net	69			
6.16	create logic port	70			
6.17	create power domain	71			
6.18	create power switch				
6 1 9	create_pst [legacy]	80			
6 20	create supply net	80			
6.21	create supply_net	83			
6.21	create supply_pole	05			
6.22	create unf?hdl vct	04			
6.23	describe state transition	05			
6.25	and nower model	80			
6.26	find_philasta	07			
6.20	land simplets helping	00			
0.27		90			
0.28	load_upi	91			
6.29		92			
6.30	map_isolation_cell [deprecated]	93			
6.31	map_level_shifter_cell [deprecated]	93			
6.32	map_power_switch	93			
6.33	map_retention_cell	94			
6.34	merge_power_domains [deprecated]	97			
6.35	name_format	98			
6.36	save_upf	99			
6.37	set_design_attributes	100			
6.38	set_design_top	101			
6.39	set_domain_supply_net [legacy]	101			
6.40	set_equivalent	102			
6.41	set_isolation	104			
6.42	set_isolation_control [deprecated]	110			
6.43	set_level_shifter	111			
6.44	set_partial_on_translation	116			
6.45	set_pin_related_supply [deprecated]	116			
6.46	set_port_attributes	117			
6.47	set power switch [deprecated]	121			
6.48	set repeater	121			
6.49	set retention	124			
6.50	set retention control [deprecated]	128			
6.51	set retention elements	128			
6.52	set scope	129			
6.53	set simstate behavior	130			
6.54	upf version	131			
6.55	use interface cell	132			
0.00					
Power management cell commands					
71	Introduction	135			
7 2	define always on cell	136			
73	define_diode_clamp	137			
1.5	demo_doue_oump	1.57			

7.

	7.4	define_isolation_cell	. 138	
	7.5	define_level_shifter_cell	. 141	
	7.6	define_power_switch_cell	. 145	
	7.7	define_retention_cell	. 147	
8.	UPF]	processing	. 150	
	81	Overview	150	
	8.2	Data requirements	150	
	8.3	Processing phases	150	
	8.4	Error checking	. 153	
9.	Simu	lation semantics	154	
	0.1	Supply network creation	154	
	9.1	Supply network creation	155	
	93	Power state simulation	157	
	9.4	Simstate simulation	159	
	9.5	Transitioning from one simstate state to another	161	
	9.6	Simulation of retention	. 162	
	9.7	Simulation of isolation	. 168	
	9.8	Simulation of level-shifting	. 168	
	9.9	Simulation of repeater	. 168	
Annex	A (int	formative) Bibliography	169	
Annex	B (no	rmative) HDL package UPF	170	
Annex C (normative) Queries				
Annex D (informative) Replacing deprecated and legacy commands and options				
Annex	E (inf	ormative) Low-power design methodology	. 227	
Annex	F (no	rmative) Value conversion tables	. 252	
Annex	G (no	rmative) Supporting hard IP	255	
Annex	H (no	rmative) UPF power-management commands semantics and Liberty mappings	258	
Annex I (informative) Power-management cell modeling examples				
Annex	J (info	ormative) Switching Activity Interchange Format	. 303	

IEEE Standard for Design and Verification of Low-Power Integrated Circuits

IMPORTANT NOTICE: This standard is not intended to ensure safety, security, health, or environmental protection in all circumstances. Implementers of the standard are responsible for determining appropriate safety, security, environmental, and health practices or regulatory requirements.

This IEEE document is made available for use subject to important notices and legal disclaimers. These notices and disclaimers appear in all publications containing this document and may be found under the heading "Important Notice" or "Important Notices and Disclaimers Concerning IEEE Documents." They can also be obtained on request from IEEE or viewed at http://standards.ieee.org/IPR/ disclaimers.html.

1. Overview

1.1 Scope

This standard establishes a format used to define the low-power design intent for electronic systems and electronic intellectual property (IP). The format provides the ability to specify the supply network, switches, isolation, retention, and other aspects relevant to power management of an electronic system. The standard defines the relationship between the low-power design specification and the logic design specification captured via other formats [e.g., standard hardware description languages (HDLs)].

1.2 Purpose

The standard provides portability of low-power design specifications that can be used with a variety of commercial products throughout an electronic system design, analysis, verification, and implementation flow.

1.3 Key characteristics of the Unified Power Format

The Unified Power Format (UPF) provides the ability for electronic systems to be designed with power as a key consideration early in the process. UPF accomplishes this by allowing the specification of what was traditionally physical implementation-based power information early in the design process—at the register transfer level (RTL) or earlier. Figure 1 shows UPF supporting the entire design flow. UPF provides a

consistent format to specify power design information that may not be easily specifiable in an HDL or when it is undesirable to directly specify the power semantics in an HDL, as doing so would tie the logic specification directly to a constrained power implementation. UPF specifies a set of HDL attributes and HDL packages to facilitate the expression of power intent in HDL when appropriate (see <u>Table 4</u> and <u>Annex B</u>). UPF also defines consistent semantics across verification and implementation, i.e., what is implemented is the same as what has been verified.



Figure 1—UPF tool flow

As indicated in Figure 1, UPF files are part of the design source and, when combined with the HDL, represent a complete design description: the HDL describing the logical intent and the UPF describing the power intent. Combined with the HDL, the UPF files are used to describe the intent of the designer. This collection of source files is the input to several tools, e.g., simulation tools, synthesis tools, and formal verification tools. UPF supports the successive refinement methodology (see <u>4.8</u>) where power intent information will grow along the design flow to provide needed information for each design stage.

- Simulation tools can read the HDL/UPF design input files and perform RTL power-aware simulation. At this stage, the UPF may only contain abstract models such as power domains and supply sets without the need to create the power and ground network and implementation details.
- Synthesis tools can read the HDL/UPF design input files and produce a netlist. The tool or user may
 produce a new UPF fileset that, combined with the netlist, represents a further refined version of
 same design.
- In those cases where design object names change, a UPF file with the new names is needed. A UPF-aware logical equivalence checker can read the full design and UPF filesets and perform the checks to ensure power-aware equivalence.
- Place and route tools read both the netlist and the UPF files and produce a physical netlist, potentially including an output UPF file.

IEEE Std 1801-2013 IEEE STANDARD FOR DESIGN AND VERIFICATION OF LOW-POWER INTEGRATED CIRCUITS

UPF is a concise power intent specification capability. Power intent can be easily specified over many elements in the design. A UPF specification can be included with the other deliverables of IP blocks and reused along with the other delivered IP. UPF supports various methodologies through carefully defined semantics, flexibility in specification, and, when needed, defined rational limitations that facilitate automation in verification and implementation (see Annex E).

A *UPF specification* defines how to create a supply network to supply power to each instance, how the individual supply nets behave with respect to one another, and how the logic functionality is extended to support dynamic power switching to these logic instances. By controlling the states and voltages of the supplies provided to the supply network, and by controlling the states of power switches that are part of the supply network, the power management logic of a system can cause each functional region to receive the power required to complete its computational tasks in a timely manner.

1.4 Use of color in this standard

This standard uses a minimal amount of color to enhance readability. The coloring is not essential and does not affect the accuracy of this standard when viewed in pure black and white. The places where color is used are the following:

- Cross references that are hyperlinked to other portions of this standard are shown in <u>underlined-blue</u> <u>text</u> (hyperlinking works when this standard is viewed interactively as a PDF file).
- Syntactic keywords and tokens in the formal language definitions are shown in **boldface-red text**.
- Command arguments that can be provided incrementally (*layered*) are shown in **boldface-green** text. See also <u>5.11</u>.
- Syntactic keywords and tokens that have been explicitly identified as legacy or deprecated constructs (see <u>6.1</u>) may be shown in **brown text**.

1.5 Contents of this standard

The organization of the remainder of this standard is as follows:

- <u>Clause 2</u> provides references to other applicable standards that are presumed or required for this standard.
- <u>Clause 3</u> defines terms and acronyms used throughout the different specifications contained in this standard.
- <u>Clause 4</u> describes the basic concepts underlying UPF.
- <u>Clause 5</u> describes the language basics for UPF and its commands.
- <u>Clause 6</u> details the syntax and semantics for each UPF power intent command.
- <u>Clause 7</u> details the syntax and semantics for each UPF power-management cell command.
- <u>Clause 8</u> defines a reference model for UPF command processing.
- <u>Clause 9</u> defines simulation semantics for various UPF commands.
- Annexes. Following <u>Clause 9</u> are a series of annexes.

2. Normative references

The following referenced documents are indispensable for the application of this standard (i.e., they must be understood and used, so each referenced document is cited in the text and its relationship to this document is explained). For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

IEC 61691-1-1/IEEE Std 1076TM, Behavioural languages—Part 1: VHDL Language Reference Manual.^{1, 2}

IEEE Std 1800TM, IEEE Standard for SystemVerilog—Unified Hardware Design, Specification, and Verification Language.³

3. Definitions, acronyms, and abbreviations

For the purposes of this document, the following terms and definitions apply. The *IEEE Standards Dictionary Online* [B1] should be consulted for terms not defined in this clause. ^{4, 5} Certain terms in this standard reflect their corresponding definitions in IEEE Std 1800 or IEC 61691-1-1/IEEE Std 1076, or they are listed in Annex $\underline{A}^{.6}$

3.1 Definitions

active component: A **component** that contains one or more input receivers and one or more output drivers whose values are functions of the inputs, but whose inputs and outputs are not directly connected; or any HDL construct(s) that synthesize(s) to an active component.

active control signal: A control signal that is currently presenting the value (level) or transition (edge) that enables or triggers an active component to operate in a particular manner.

active power state: A power state whose logic expression and, if present, supply expression evaluate to *True* at a given time.

activity: Any change in the value of a net, regardless of whether that change is propagated to an output.

ancestor: Any instance between the current scope in the logic hierarchy and its root scope. When the current scope is a top-level module, it does not have any ancestors. *See also:* descendant.

anonymous object: An object that is not named in the context of UPF. Implementations may assign a legal name, but such names are not visible in the UPF context.

balloon latch: A retention element style in which a register's value is saved to a dedicated latch at power-down and the latch value is restored to the register at power-up.

boundary instance: An instance that has no parent or whose parent is in a different power domain.

- ¹IEC publications are available from the International Electrotechnical Commission (http://www.iec.ch/). IEC publications are also available in the United States from the American National Standards Institute (http://www.ansi.org/).
- ²IEEE publications are available from The Institute of Electrical and Electronics Engineers (http://standards.ieee.org/).

³The IEEE standards or products referred to in this clause are trademarks of The Institute of Electrical and Electronics Engineers, Inc. ⁴*IEEE Standards Dictionary Online* subscription is available at:

 $[\]underline{http://www.ieee.org/portal/innovate/products/standard/standards_dictionary.html.}$

⁵The numbers in brackets correspond to those of the bibliography in <u>Annex A</u>.

⁶Information on references can be found in <u>Clause 2</u>.

IEEE Std 1801-2013 IEEE STANDARD FOR DESIGN AND VERIFICATION OF LOW-POWER INTEGRATED CIRCUITS

component: A physical and logical construction that relates inputs to outputs.

composite domain: A power domain consisting of subordinate **power domains** called **subdomains**. All **subdomains** in a composite domain share the same **primary supply set**. Any operation performed on a composite domain has the same effect as performing the operation on each of its **subdomains**.

configuration UPF: The UPF specification for an intellectual property (IP) block that defines a particular configuration of the block for use in a given system. The configuration UPF typically includes the **constraint UPF** and extends it with configuration-specific details. Sometimes referred to as **golden UPF**.

connected: Attached together via a direct connection.

constraint UPF: The UPF specification for an intellectual property (IP) block that defines constraints that must be met by any configuration of the IP block used in a larger system. Sometimes referred to as **platinum UPF**.

corruption semantics: The rules defining the behavior of logic response to reduction or disconnection of power to that logic.

current scope: The design hierarchy location that serves as the immediate context for interpretation and execution of UPF commands. Also, the **instance** specified by the **set_scope** command.

NOTE—See <u>6.52</u>.⁷

declared: Specified in the HDL explicitly or implicitly via a UPF command.

descendant: Any **instance** between the **current scope** in the **logic hierarchy** and its **leaf-level instance**. When the **current scope** is a **leaf-level instance**, it does not have any descendants. *See also:* **ancestor**.

descendant subtree: A portion of a **logic hierarchy**, rooted at one **instance** in the hierarchy, and containing that **instance** and all of its **descendants**.

design hierarchy: A hierarchical structure of nested definitions described in an HDL.

direct connection: A physical wire; or any HDL construct(s) that synthesize(s) to a direct connection.

domain port: A port that is on the interface of a power domain.

driver: The source or drain of a transistor, if the drain or source is connected to a power rail; a complementary metal oxide semiconductor (CMOS) inverter that continually connects a node to power or ground; any component that sets the value of its output via a transistor or inverter; a constant assignment; any combinational logic including a buffer of any kind; any sequential logic; or any HDL construct(s) that synthesize(s) to such combinational or sequential logic.

driver supply: For a **driver** that is a transistor, the supply connected to its source or drain; for a **driver** that is an inverter, the pair of supplies connected to the source/drain of the transistor pair comprising the inverter; or for an output of an **active component**, the related **supply set** of that output.

electrically equivalent: For **supply port**s/**net**s, connected (whether the connections are evident or not in the design) without any intervening switches, and therefore guaranteed to have the same value at all times from the perspective of any load; for **supply set**s/set handles, consisting of a set of electrically equivalent **supply net**s for each required function.

⁷Notes in text, tables, and figures are given for information only and do not contain requirements needed to implement the standard.

equivalent: A pair of supply nets or a pair of supply sets that are considered to be interchangeable for certain purposes. *See also:* electrically equivalent; functionally equivalent.

erroneous: A usage that is likely to lead to an error in the design, but that tools may not be able to detect and report.

extent (of a domain): The set of instances that comprise a power domain.

feedthrough: A direct connection between two **ports** on the **interface of a power domain**, where the connection involves two **ports** on the upper boundary, or two **ports** on the lower boundary, or one of each; also, a direction connection between two **ports** of the same **leaf-level instance**.

feedthrough port: A port on the interface of a power domain that is part of a feedthrough through that domain, or a port on the interface of a leaf-level instance that is part of a feedthrough through that instance.

functionally equivalent: Functioning identically from the perspective of any load, either as a result of being **electrically equivalent**, or due to independent but parallel circuitry.

generate block: In the HDL code, this represents a level of design hierarchy, although a generate block is not itself an **instance**. After synthesis, generate blocks do not exist as an independent level of hierarchy. It is illegal to create any UPF objects in a **scope** that corresponds to a generate block.

golden source: The design together with the constraint UPF and the configuration UPF.

golden UPF: See: configuration UPF.

hard macro: A block that has been completely implemented and can be used as it is in other blocks. This may be modeled by an HDL module for verification or as a library cell for implementation.

hierarchical name: A series of names separated by the **hierarchical separator character**, the final name of which may be any legal HDL name or UPF name, and each preceding name is the name of an **instance** or **generate block** in which the following name is declared. *See also:* **hierarchical separator character**.

hierarchical separator character: A special character used in composing hierarchical names. The hierarchical separator character is a slash (/).

HighConn: The side of a **port** connection that is higher in the **design hierarchy**; the actual signal associated with a formal **port** definition.

implementation UPF: The UPF specification of how power distribution and control is to be implemented for a system. The implementation UPF typically includes the configuration UPF for each of the intellectual property (IP) blocks instantiated in the system. Sometimes referred to as **silicon UPF**.

inactive: A normally active component in a state in which it does not respond to **activity** on its inputs. Also, a control signal that is not currently presenting the value (level) or transition (edge) that enables or triggers an **active component** to operate in a particular manner.

instance: A particular occurrence of a SystemVerilog module (see IEEE Std 1800), VHDL entity (see IEC 61691-1-1/IEEE Std 1076), or library cell at a specific location within the design hierarchy.

interface of a power domain: The union of the upper boundary and the lower boundary of the power domain.

isolation: A technique used to provide defined behavior of a logic signal when its driving logic is not active.

isolation cell: An **instance** that passes logic values during normal mode operation and clamps its output to some specified logic value when a control signal is asserted.

leaf-level cell: An **instance** that has no **descendants**, or an **instance** that has the attribute **UPF_is_leaf_cell** associated with it.

NOTE—See <u>Table 4</u>.

leaf-level instance: See: leaf-level cell.

level-shifter: An **instance** that translates signal values from an input voltage swing to a different output voltage swing.

live slave: A retention element style in which the slave latch of a master-slave flip-flop (MSFF) is always on and therefore maintains the value of the MSFF during power-down.

logic hierarchy: An abstract view of a design hierarchy in which only those definitions representing instances are included.

LowConn: The side of a port connection that is lower in the design hierarchy; the formal port definition.

lower boundary (of a power domain): The HighConn side of each **port** of each **boundary instance** in the **extent** of another **power domain** whose **parent** is in the **extent** of this domain, together with the HighConn side of each **port** of any macro cell instance in this **power domain**, for which the related supply set is neither identical to nor equivalent to the **primary supply set** of this domain.

map: Identifies a specific **model** corresponding to an abstract behavior. An **instance** of the **model** can then be used to implement the specific behavior.

model: A SystemVerilog module, VHDL entity/architecture, or Liberty cell.

named power state: A **power state** defined using **add_power_state**, **add_port_state**, or **add_pst_state** for a **supply set** or **power domain**, or the **DEFAULT_NORMAL** and **DEFAULT_CORRUPT** power states predefined for supply sets.

net: The individual **net segments** that make up a collection of interconnections between a collection of **ports**. A **net** may be named or anonymous.

net segment: A direct connection within a single instance.

parent: The immediate ancestor of a given instance within the logic hierarchy.

passive component: A direct connection; a **component** that has neither a **receiver** nor a **driver**, whose output is connected to its input, and therefore its output is always the same as its input, e.g., a pass transistor; or any HDL construct(s) that synthesize(s) to a **feedthrough** component.

pg_type: An attribute of a port that indicates its use in providing power to a cell.

platinum UPF: See: constraint UPF.

port: A connection on the interface of a SystemVerilog module or VHDL entity. Also, a port on the interface of a power domain.

power domain: A collection of **instance**s that are treated as a group for power-management purposes. The **instance**s of a power domain typically, but do not always, share a **primary supply set**. A power domain may also have additional supplies, including **retention** and **isolation** supplies.

power rail: The physical implementation of a power supply net.

power state: The state of a supply net, supply port, supply set, or power domain.

NOTE—See <u>Clause 4</u>.

power state table (PST): A table that captures the legal combinations of **power states** for a set of **supply ports** and/or **supply nets**.

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

receiver: The gate of a transistor; the input to an inverter; any **component** whose behavior is determined by an input signal; any combinational logic including a buffer of any kind; any sequential logic; or any HDL construct(s) that synthesize(s) to such combinational or sequential logic.

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

regulator: An **instance** that takes a set of input **supply net**s and provides the source for a set of output **supply net**s. The output voltage is a function of the input voltages and the logical state of any control signals.

retention: Enhanced functionality associated with selected **sequential elements** or a memory such that memory values can be preserved during the power-down state of the primary supplies.

retention register: A register that extends the functionality of a **sequential element** with the ability to retain its memory value during the power-down state.

rooted name: The **hierarchical name**, relative to the **current scope**, of an object in the **logic hierarchy** or a UPF object defined for a **scope** in the hierarchy.

root scope: The topmost scope in the **logic hierarchy**, which contains an implicit instance of each top-level module.

root supply driver: The origin of a supply, e.g., an on-system voltage regulator, bias generator modeled in HDL, or an off-chip supply source. *See also:* **supply source**.

root supply source: An input or inout **supply port** that is not connected to an "upstream" **supply net**; an input or inout **supply port** that is not connected to a root supply source defined in an ancestor scope; an output or inout **supply port** that is not connected to a **supply source** defined in a child scope; a **supply set** or supply set handle function that is neither associated with a **supply port** or **supply net** (via **associate_supply_set**) nor connected to another root supply source (via **connect_supply_net**).

NOTE—See <u>6.7</u> and <u>6.11</u>.

scope: An instance in the logic hierarchy.

IEEE Std 1801-2013 IEEE STANDARD FOR DESIGN AND VERIFICATION OF LOW-POWER INTEGRATED CIRCUITS

silicon UPF: See: implementation UPF.

simple name: An identifier that denotes an object declared in a given scope and is not a hierarchical name.

simstate: The level of operational capability supported by a given power state of a supply set.

sink: A receiver; the HighConn of an input port or inout port of an instance; or the LowConn of an output port or inout port of an instance.

source: A **driver**; the LowConn of an input port or inout port of an **instance**; or the HighConn of an output port or inout port of an **instance**.

state element: A sequential element such as a flip-flop, latch, or memory element. Also, a conditionally stored value in register transfer level (RTL) code from which a sequential element would be inferred.

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

subdomain: A member of the set of domains comprising a composite power domain.

supply function: An abstraction of a **supply net** in a **supply set**, the name of which identifies the purpose of the corresponding **net** in the **supply set**.

supply net: An HDL representation of a power rail.

supply port: A connection point for supply nets.

supply set: A collection of supply functions that in aggregate provide a complete power source.

supply source: A supply port that propagates but does not originate a supply value.

switch: An **instance** that conditionally connects one or more input **supply net**s to a single output **supply net** according to the logical state of one or more control inputs.

top-level instance: An implicit instance corresponding to a top-level module.

upper boundary (of a power domain): The LowConn side of each port of each boundary instance in the extent of this power domain.

3.2 Acronyms and abbreviations

- CMOS complementary metal oxide semiconductor
- DFT Design for Test
- EDA electronic design automation
- HDL hardware description language
- IP intellectual property
- MSFF master-slave flip-flop

IEEE Std 1801-2013 IEEE STANDARD FOR DESIGN AND VERIFICATION OF LOW-POWER INTEGRATED CIRCUITS

- NMOS N-channel metal oxide semiconductor
- PG power/ground
- PMOS P-channel metal oxide semiconductor
- PST power state table
- ROM Read-only Memory
- RTL register transfer level
- SAIF Switching Activity Interchange Format
- SoC System On Chip
- Tcl Tool Command Language
- UPF Unified Power Format
- VCT value conversion table
- VHDL VHSIC hardware description language
- VHSIC Very High Speed Integrated Circuit

4. UPF concepts

This clause provides an overview of concepts involved in defining power intent using UPF. These concepts include those related to the representation of the design structure and functionality in one or more hardware description languages (HDLs), as well as those related to power-management structures and functionality defined for and/or added to the design to model intended power-management capabilities.

The structure and functionality of a design is specified using HDLs such as Verilog [B2], SystemVerilog, or VHDL. Each HDL may have specific terminology and concepts that are unique to that language, but all HDLs share some common concepts and capabilities. A typical design may be expressed in one or more HDLs.

UPF is defined in terms of a generalized abstraction of an HDL-based design hierarchy. This abstraction enables the UPF definition to apply to a design expressed in any of the three HDLs previously mentioned, or in any combination thereof, while at the same time minimizing the complexity of the UPF definition. This clause presents the abstract model and maps it to specific HDL concepts.

UPF is intended to apply to a design as its representation changes from an abstract functional model to a concrete physical model, during which process the power intent expressed in UPF becomes realized as part of the implementation. Because of this, the abstract logic hierarchy that is the basis of the UPF definition shall be understood in terms of both functional specification and physical implementation.

4.1 Design structure

4.1.1 Transistors

At the lowest level, UPF focuses on controlling power (or more precisely, voltage and current) delivered to transistors. These are usually assumed to be digital complementary metal oxide semiconductor (CMOS) transistors, but they could be analog devices as well or implemented in other technologies. The gate connection of a transistor is a receiver; the source of the signal provided to a gate (in CMOS, typically the output of a P/N transistor pair) is a driver.

4.1.2 Standard cells

Transistors are seldom modeled individually in an HDL description; typically, collections of transistors are represented by standard cells that have been developed as part of a particular technology library, which is usually expressed in the Liberty library format (see [B7]). Such cells typically have a primary supply (power and ground) and may also have a secondary supply for related behavior (e.g., state retention).

4.1.3 Hard macros

A library may also contain hard macros, which provide predefined physical implementations for much larger and more complex functions. A hard macro may have multiple supplies.

4.2 Design representation

4.2.1 Models

Library elements have corresponding behavioral models for use in simulation. These models may or may not include power and ground pins for their supplies. Standard cell models are usually written as Verilog modules and use constructs such as Verilog built-in primitives or user-defined primitives (UDPs) to express the relatively simple behavior of a standard cell. They may also be written as VHDL design entities (entity/

architecture pairs) using package VITAL, which provides Verilog-like primitive modeling capabilities. Hard macro models may be written in either language, using more complex behavioral constructs such as Verilog initial blocks and always blocks or VHDL processes and concurrent statements.

4.2.2 Netlist

A netlist is a collection of unique instances of standard cells and hard macros, interconnected by nets (Verilog) or signals (VHDL). Such instances are considered to be leaf-level instances, because their models are not constructed from an interconnection of subordinate instances, but instead are built using behavioral or functional HDL statements. A netlist may also include hierarchical instances, i.e., instances of a model that is itself defined as a netlist.

A power/ground (PG) netlist is a netlist containing cell and/or hard macro instances that include power and ground pins and a representation of the power and ground supply routing for those instances. A non-PG netlist is one that does not include any representation of the power supply network.

4.2.3 Behavioral models

Behavioral models that are written using the RTL synthesis subset of Verilog or VHDL are synthesizable models, or *soft macros*, which can be read by an RTL synthesis tool and mapped to a functionally equivalent netlist. Synthesis involves identifying or inferring the state elements needed to implement the specified behavior and implementing the combinational logic interconnecting those elements and the model's ports.

For many synthesizable HDL constructs, synthesis creates combinational or sequential logic elements that are ultimately defined in terms of transistors, which in turn define drivers and receivers. In particular, any synthesizable statement that involves conditional computation or conditional updating of an output will most likely create logic. In contrast, unconditional assignment statements and port associations typically result in interconnect, not logic; for such HDL constructs, no drivers or receivers are created. In particular, ports do not create drivers; it is the logic driving a port that creates a driver for the port and for the net associated with the port.

4.2.4 HDL scopes

An HDL model defines one or more scopes. A scope is a region of HDL text within which names can be defined. Such names are typically visible (i.e., can be referenced) within the scope in which they are defined and, in certain cases, in other scopes (e.g., nested scopes). A Verilog model usually defines a single scope for the whole model. A VHDL model often defines multiple scopes; one for the whole model, plus other nested scopes for process statements and block statements. generate statements in either HDL are also considered to be nested scopes within the model's top-level scope.

4.2.5 Design hierarchy

A design hierarchy is constructed by defining one model in terms of interconnected instances of other models. Each instance represents a subtree of the hierarchy; the boundary between this subtree and its parent instance is defined by the interface of the model that has been instantiated to create the subtree. The interface consists of the model's ports, together with the nets associated with those ports for the instance that created this subtree. In Verilog, a port is defined as having two sides: a *HighConn* and a *LowConn*. The LowConn represents the port declaration in the model; the HighConn represents an instance of that port associated with an instance of the model, and therefore indirectly the net attached to that port instance. In VHDL, a somewhat different distinction is made between a "formal" port of a model and the "actual" signal associated with that port for a given instance of the model. In the context of UPF, regardless of what HDL is involved, the term LowConn means the (formal) port declaration in the model and the term HighConn means the port of an instance of a model and by extension the net or signal connected to that port.

An HDL model that is not instantiated in any other instance is a top model, or simply top. A given design hierarchy usually contains a single top, but it may contain multiple tops in certain cases (e.g., if the design and the testbench in a simulation are modeled separately—neither instantiates the other). Each top is considered to be implicitly instantiated within the *root scope*. In Verilog, the root scope is \$root; in VHDL, the root scope is the *root declarative region*. The instance name of such an implicit instance is the same as the model name.

4.2.6 Logic hierarchy

UPF assumes a somewhat more abstract model of the design hierarchy. This abstract model is called the *logic hierarchy*. As usual, the topmost scope is still the root scope and modules that are not instantiated elsewhere are the top modules (and instances) of the hierarchy. However, in the logic hierarchy, each scope corresponds to a whole instance; internal scopes presented in the design hierarchy are not modeled. In particular, HDL generate statements, which are considered to be internal scopes in the respective language definitions, are assumed to be collapsed into the parent module scope in the logic hierarchy.

UPF generally allows references to the names of objects defined anywhere in the subtree descending from a given instance, when the *current scope* is set to that instance. Such references are called *rooted names*, meaning they are hierarchical names relative to the current scope. If the design hierarchy contains generate statements that have been collapsed in the logic hierarchy, then the hierarchical name of an object in the logic hierarchy may include simple names that encode the collapsed scope names.

UPF also uses the logic hierarchy as a framework for locating the power-management objects used to represent power-management concepts, e.g., power domains and power state tables (PSTs). Each such object is effectively declared in a specific scope of the logic hierarchy, and the name of the scope can be used as the prefix of the name of the object.

The logic hierarchy can be viewed as a purely conceptual structure that is independent of the eventual physical implementation. Alternatively, the logic hierarchy can be viewed as an indication of the floor plan to be used in the physical implementation. Either view can be used, but it is best to adopt one view or the other for a given design, because the choice can affect how the power intent is expressed in UPF.

4.2.7 Hierarchy navigation

In UPF, commands are executed in the context of a scope within the logic hierarchy. The set_scope command (see 6.52) is used to navigate within the hierarchy and to set the current scope within which commands are executed.

Consistent with SystemVerilog \$root, the root of the logic hierarchy is the scope in which the top modules are implicitly instantiated. Other locations within the logic hierarchy are referred to as the *design top instance*, which has a corresponding *design top module*, and the current scope.

The design top instance and design top module are typically paired: the design top instance (represented by a hierarchical name relative to the root scope) is an instance in the hierarchy representing a design for which power intent has been defined, and the design top module is the module for which the UPF file expressing this power intent has been written. The association between the UPF file and the design top module is specified in the UPF file using **set_design_top** (see <u>6.38</u>); this UPF file is then typically applied to each instance of that module in a larger system.

The current scope is an instance that is, or is a descendant of, the design top instance (represented by a relative path name from the design top instance).

The set_scope command (see 6.52) changes the current scope locally within the subtree depending on the current design top instance/module. Since the design top instance is typically an instance of the design top

module, they both have the same hierarchical substructure; therefore, **set_scope** can be written relative to the module, but still work correctly when applied to an instance. The **set_scope** command is only allowed to change scope within this subtree. It cannot scope above the current design top instance.

The design top instance and design top module are initially set by the tool, possibly with direction from the user. They can be changed by invoking **load_upf** with the **-scope** argument (see 6.28). The current scope is reset whenever the design top instance changes. When **load_upf** completes, all three variables revert back to their previous settings.

4.2.8 Ports and nets

Ports define connection points between adjacent levels of hierarchy. In HDL, ports are defined as part of the interface of a module and therefore exist for each instance of the module. Nets define interconnections between a collection of ports. In HDL, nets are defined within a module and therefore exist within each instance of the module.

A port has two sides. The top side is the HighConn side, which is visible to the parent of the instance whose interface contains the port. The bottom side is the LowConn side, which is visible internal to the instance whose interface contains the port.

When a net in the current scope is connected to a port on a child instance, the connection is made to the HighConn side of the port. When a net in the current scope is connected to a port defined on the interface of the instance that is the current scope, the connection is made to the LowConn side of the port.

A port can be referenced wherever a net is required. Such a reference refers to the LowConn side of the port. A port can be thought of as being implicitly connected to an implicit net created with the same name and in the same scope as the LowConn side of the port.

4.2.9 Connecting nets to ports

In an HDL description, ports are typically required to pass nets from one level of hierarchy to another. In UPF, a net in the current scope can be connected to the LowConn of any port declared in the same scope or to the HighConn of any port within its descendant subtree. If the port is not declared in the same scope as the net, additional ports, nets, and port/net associations may be created to establish the connection from the net to the port. Such implicitly created ports and nets shall have the same simple name as the net being connected unless that name conflicts with the name of an existing port or net; in which case, to avoid a name conflict, the tool shall create a name that is unique for that scope.

NOTE—Nets are propagated as necessary through the descendant subtree and may be renamed to avoid name collision; therefore, the same simple name in different scopes may refer to nets that are independent and unconnected.

Implicitly created ports and nets should not be referenced directly by UPF commands, since the names of such ports and nets are not guaranteed to be the same as the original net name. These implicitly created ports and nets are merely a method of implementing a UPF connection in terms of valid HDL connections, when the UPF-specified power intent is represented in HDL form.

4.3 Power architecture

A UPF power intent specification defines the power architecture to be used in managing power distribution within a given design. The power architecture defines how the design is to be partitioned into regions that have independent power supplies, and how the interfaces between and interaction among those regions will be managed and mediated.

4.3.1 Power domains

A power domain is a collection of instances that are typically powered in the same way. In the physical implementation, the instances of a power domain are typically placed together and powered by the same power rails. In the logic hierarchy, the instances of a power domain are typically part of the same subtree of the hierarchy, or of sibling subtrees with a common ancestor, and powered by the same supply nets.

A power domain is defined within a scope (or instance) in the logic hierarchy. The definition of the power domain identifies the uppermost instances of the domain: those that define the upper boundary of the domain. For any given instance included in the power domain, a child instance of the given instance is transitively included in the power domain, unless that child instance is explicitly excluded from this power domain or is explicitly included in the definition of another power domain.

More formally, a boundary instance of a given power domain is any instance that has no parent (it is an implicit instance of a top-level module) or whose parent is in the extent of a different power domain. It is possible for one boundary instance of a power domain to be an ancestor of another boundary instance of the same power domain. This occurs when one instance is in the extent of a given power domain and both an ancestor and a descendant of that instance are in the extent of a second power domain. In this case, both the ancestor and the descendant may be boundary instances of the second domain. A domain with such a structure is referred to as a *donut* power domain.

The upper boundary of a power domain consists of the LowConn side of each port on each boundary instance in the domain. The lower boundary of a domain consists of the HighConn side of each port on each child instance that is in some other power domain or is a port of a macro cell instance that is powered differently from the rest of the domain. Both boundaries include any logic ports added to the design for power management. The interface of a power domain consists of the upper boundary and the lower boundary.

The instance in the logic hierarchy in which a power domain is defined is called the *scope* of the power domain. The set of instances that belong to a power domain are said to be the extent of that power domain. This distinction is important: while a given instance can be the scope of multiple power domains, it can be in the extent of one and only one power domain. As a consequence of these definitions, all instances within the extent of a domain are necessarily within the scope of the domain or its descendants.

A power domain can be either contiguous or non-contiguous. In the physical implementation, a contiguous power domain is one in which all instances are placed together; a non-contiguous power domain is one in which instances in the domain are placed in two or more disjoint locations. A power domain is contiguous within the logic hierarchy if it contains a single boundary instance; it is non-contiguous within the logic hierarchy if it non-contiguous instances.

For a non-contiguous power domain, a connection from an instance in the extent of the power domain to some other instance in the extent of the domain may need to be routed through another power domain.

Power domains that share a primary supply set may be composed together to form a larger power domain such that operations performed on this larger power domain apply transitively to each subdomain. In this way, unnecessary power domains may be aggregated together and handled as one for simplicity.

After UPF-specified power intent has been completely applied, it is an error if any instance is not included in a power domain.

4.3.2 Drivers, receivers, sources, and sinks

A logic signal in the design originates at an active component (the driver) and terminates at another active component (the receiver). Along the way it may pass through ports and nets. The driver and any port it

passes through on the way to a receiver is considered a source; the receiver and any port it passes through on the way from the driver is considered a sink. For example, a buffer defines both a source and a sink: the buffer's output port is a source; the buffer's input port is a sink.

A signal traversing a power domain may or may not be driven within the power domain. A port is neither a driver nor a receiver; it merely propagates a signal across a hierarchy boundary. If a port on the interface of a power domain is connected directly to another port on the interface of the same power domain, without going through an active component, the connection between those two ports has neither a driver nor a receiver in that domain. In this case, the connection is a feedthrough path through that domain.

HDL assignment statements may include delays, which may represent inertial delay (resulting from transistor switching) or transport delay (resulting from propagation along a wire). However, synthesis tools typically ignore such delays; therefore, the inclusion of such a delay, whether inertial or transport, does not by itself imply that an active component will be inferred from the assignment. For this reason, delays are not considered to create drivers or receivers.

A connection may be thought to "exist" in a given domain, if a user so chooses, but since a connection is by definition a passive component, it has no driver in the domain in which it exists and therefore is not affected or corrupted by the power state of the domain in which it exists.

4.3.3 Isolation and level-shifting

Two power domains interact if one contains logic that is the driver of a net and the other contains logic that is a receiver of the same net. When both power domains are powered up, the receiving logic should always see the driving logic's output as an unambiguous 1 or 0 value, except for a very short time when the value is in transition. The structure of CMOS logic typically ensures that minimal current flow will occur when the input value to a gate is a 1 or 0. However, if the driving logic is powered down, the input to the receiving logic may float between 1 or 0. This can cause significant current to flow through the receiving logic, which can damage the circuit. An undriven input can also cause functional problems if it floats to an unintended logic value.

To avoid this problem, isolation cells are inserted at the boundary of a power domain to ensure that receiving logic always sees an unambiguous 1 or 0 value. Isolation may be inserted for an input or for an output of the power domain. An isolation cell operates in two modes: normal mode, in which it acts like a buffer, and isolation mode, in which it clamps its output to a defined value. An isolation enable signal determines the operational mode of an isolation cell at any given time.

Two interacting power domains may also be operating with different voltage ranges. In this case, a logic 1 value might be represented in the driving domain using a voltage that would not be seen as an unambiguous 1 in the receiving domain. Level-shifters are inserted at a domain boundary to translate from a lower to a higher voltage range, and sometimes from a higher to a lower voltage range as well. The translation ensures the logic value sent by the driving logic in one domain is correctly received by the receiving logic in the other domain.

Isolation and level-shifting are often implemented in combination, so one standard cell implements both functions. UPF includes support for such "combo" cells.

Isolation and level-shifter strategies specify that isolation and level-shifter cells are to be inserted in specified locations. However, there are some cases where implementation tools may choose not to insert such cells, or to optimize redundant insertion of such cells. For example, isolation/level-shifters on floating ports that appear to have no drivers or have constant drivers may be removed or transformed, provided the resulting behavior is unchanged. To prevent implementation tools from applying such optimizations, isolation and level-shifting strategies can instead specify the respective cells are to be inserted regardless of optimization possibilities.

4.3.4 State retention

State retention is the ability to retain the value of a state element in a power domain while switching off the primary power to that element, and being able to use the retained value as the functional value of the state element upon power-up. State retention can enable a power domain to return to operational mode more quickly after a power-down/power-up sequence and it can be used to maintain state values that cannot be easily recomputed on power-up. State retention can be implemented using retention memories or retention registers. Retention registers are sequential elements (latches or flip-flops) that have state retention capability.

For a retention register, the following terms apply:

- Register value is the data held in the storage element of the register. In functional mode, this value
 gets updated on the rising/falling edge of clock or gets set or cleared by set/reset signals,
 respectively.
- *Retained value* is the data in the retention element of retention register. The retention element is powered by the retention supply.
- *Output value* is the value on the output of the register.

Depending on how the retained value is stored and retrieved, there are at least two flavors of retention registers, as follows:

- a) *Balloon-style retention*: In a balloon-style retention register, the retained value is held in an additional latch, often called the *balloon latch*. In this case, the balloon element is not in the functional data-path of the register.
- b) *Master/slave-alive retention*: In a master/slave-alive retention register, the retained value is held in the master or slave latch. In this case, the retention element is in the functional data-path of the register.

A balloon-style retention register typically has additional controls to transfer data from a storage element to the balloon latch, also called the *save step*, and transfer data from the balloon latch to the storage element, also called the *restore step*. The ports to control the save/restore pins of the balloon style retention register need to be available in the design to describe and implement this style of registers.

A master/slave-alive retention register typically does not have additional save/restore controls as the storage element is the same as the retention element. Additional control(s) on the register may park the register into a quiescent state and protect some of the internal circuitry during power-down state, and thus ensure the retention state is maintained. The restore in such registers typically happens upon power-up, again owing to the storage element being the same as the retention element. Thus, this style of registers may not specify save/restore signals, but may specify a retention condition that could take the register in and out of retention.

4.4 Power distribution

The electric current transported by a supply net originates at a root supply driver, which may be an on-chip voltage regulator, a bias generator modeled in HDL, or an off-chip supply source. A root supply driver's value may be conditionally propagated by a switch (modeled in HDL or created in UPF, see 6.18).

A root supply source (see 3.1) has an implicit root supply driver associated with it. Initially, the root supply driver drives the root supply source with the value {OFF, unspecified}. The package UPF functions supply_on and supply_off may be called to change the driving value of the root supply driver that drives a root supply source. It shall be an error if either of these functions is applied to an object that is not a root supply source. A root supply driver may also be created and manipulated via functions defined in package UPF (see <u>Annex B</u>).

IEEE Std 1801-2013 IEEE STANDARD FOR DESIGN AND VERIFICATION OF LOW-POWER INTEGRATED CIRCUITS

A supply net may have one or more supply sources, depending upon its resolution type. During UPF processing, if the number of sources connected to a supply net do not conform to the requirements of its resolution type, an error shall be reported. At any given time during simulation, if the sources of a supply net do not conform to the requirements of its resolution type, the resolved value of the supply net at that time is set to {UNDETERMINED, unspecified}.

A power switch may have one or more input supply ports and one output supply port. Each input supply port may have one or more state definitions. At any given time during simulation, if the state definitions of a given input supply port are contradictory, or if multiple incompatible inputs are enabled at the same time, or if any input supply port is in an error state, the resolved value of the output supply port at that time is set to {UNDETERMINED, unspecified}.

The semantics defined in this standard, such as the supply net resolution functions, presume an idealized supply network with no voltage drop; the semantics for supply network resolution with modeled-voltage drop are outside the scope of this standard.

4.4.1 Supply network elements

Supply network objects (supply ports, supply nets, and switches) are created within the logic hierarchy to provide connection points for a root supply and to propagate the value of a root supply throughout a portion of the design. Supply network objects are created independent of power-domain definitions. This allows sharing of common components of the supply distribution network across multiple power domains.

4.4.1.1 Supply ports and nets

Supply ports provide a connection point for supply nets where they cross a hierarchy boundary. Supply nets can be used to create a connection between two supply ports or from a supply port to an instance within a power domain.

Supply ports and nets may be created in UPF or in the HDL design. If created in the HDL, the port or net shall be of the supply net type defined in the appropriate package UPF (see <u>Annex B</u>).

4.4.1.2 Supply switches

Supply switches conditionally propagate the value on an input supply port to an output supply port, depending upon the value of a control signal. A supply net may be connected to one or more power switches or supply ports, which may be connected to one or more root supply drivers.

4.4.1.3 Supply sets

A supply set represents a collection of supply nets that provide a complete power source for one or more instances. Each supply set defines six standard functions: **power**, **ground**, **pwell**, **nwell**, **deeppwell**, and **deepnwell**. Each function represents a potential supply net connection to a corresponding portion of a transistor. Each function of a given supply set can be associated with a particular supply net that implements the function.

A global supply set is one that is defined in a given scope and associates supply nets with its functions. One or more local supply sets, called *supply set handles*, can be defined for a power domain, a power switch, an isolation strategy (see 6.41), a level-shifting strategy (see 6.43), or a retention strategy (see 6.49). A supply set can be associated with a supply set handle as a whole; the functions of a supply set handle can be broken out and connected to ports of instances. This association creates a connection between the supply nets represented by corresponding functions of the supply set and supply set handle.

A supply set function is equivalent to a supply net and may be used anywhere a supply net is allowed. The supply set function represents the supply net that is or will be associated with that function of the supply set. The supply set function reference is a symbolic name for the supply net it represents.

A reference to a supply net by its symbolic name is an indirect reference.

NOTE—A supply net may be associated with a function of more than one supply set. The function that a given supply net performs in one supply set is unrelated to the function it may perform in any other supply set.

4.4.2 Supply network construction

Supply ports and nets are interconnected to create a supply network. Certain definitions and restrictions constrain how these interconnections are made.

4.4.2.1 Supply sources and loads

Supply ports define supply sources and supply loads, as follows:

- The LowConn of an input or inout port is a supply source. The HighConn of an output or inout port is a supply source (including a switch output).
- The LowConn of an output or inout port is a load. The HighConn of an input or inout port is a load (including a switch input).

A port that is neither a top-level port nor a leaf-level port is an internal (hierarchical) port.

4.4.2.2 Supply port/net connections

Connections are made from nets to ports

- a) from a net to (the LowConn of) a port declared in the same scope; or
- b) from a net to (the HighConn of) a port declared in a lower scope; or
- c) from a net to a pin of a leaf cell.

The LowConn of a port may be used as an implicit net and connected to another port.

Only one net connection can be made to the LowConn of a port. Likewise, only one net connection can be made to the HighConn of a port. A source can be connected to a net that is in turn connected to multiple loads.

4.4.2.3 Supply net resolution

A supply net may be unresolved or resolved, as follows:

- An unresolved supply net shall have only one supply source connection.
- A resolved supply net may have multiple supply source connections. The resolution type may restrict how many supply sources can be on at the same time.

A supply net may have any number of load connections.

4.4.2.4 Supply net / supply set connections

Related supply nets can be grouped into a supply set, with each supply net in the group providing one or more functions of the supply set. The supply net corresponding to a given function of a supply set can be specified when the supply set is created or updated (see <u>6.22</u>). One supply set may be associated with another supply set (see <u>6.7</u>); this implicitly connects corresponding functions together and therefore it also

implicitly connects the supply nets associated with corresponding functions and any instance ports to which those functions are connected.

4.4.2.5 Supply set function connections

Supply functions of a supply set, and the supply nets they represent, can be connected to instances in one of the following ways: explicitly, automatically, or implicitly. Connections are made downward, from ports or nets in the current scope to ports of descendant instances that are in the extent of the domain.

4.4.2.5.1 Explicit and automatic connections

An explicit connection connects a given particular supply set function directly to a specified supply port. See also 6.11 and 6.12.

An automatic connection connects each supply set function to ports of selected instances, based on the pg_type of each port, as indicated by the UPF_pg_type attribute (see <u>6.46</u>) or the Liberty pg_type attribute.

For automatic connections, the default connection semantics for each function of a supply set are as follows:

- a) **power** is connected by default to ports having the *pg_type* primary_power.
- b) ground is connected by default to ports having the *pg_type* primary_ground.
- c) **pwell** is connected by default to ports having the *pg_type* pwell.
- d) **nwell** is connected by default to ports having the *pg_type* nwell.
- e) **deeppwell** is connected by default to ports having the *pg_type* deeppwell.
- f) **deepnwell** is connected by default to ports having the *pg_type* deepnwell.

4.4.2.5.2 Implicit connections

An implicit connection connects the power and ground functions of a supply set to cell instances that do not have explicit supply ports. Such connections may involve implicit creation of ports and nets, as described in 4.2.9.

Implicit supply set connections are made in each of the following cases:

a) Primary supply set

The functions of a domain's primary supply set are implicitly connected to any instance in the extent of the domain if the instance has no supply ports defined on its interface.

b) Retention supply set

The functions of a retention strategy's supply set are implicitly connected to the state element that implements retention functionality (e.g., a balloon latch, shadow register, or live slave latch) for any register in the domain to which the strategy applies.

c) Isolation supply set

The functions of a supply set for an isolation strategy are implicitly connected to the corresponding isolation cell implied by the application of the strategy.

d) Level-shifter supply sets

The functions of a supply set for a level-shifting strategy are implicitly connected as appropriate to the input, output, or internal supply pins of any level-shifter implied by the application of the strategy.
After UPF-specified power intent has been completely applied, it shall be an error if any instance in the design does not have a supply set function or supply net connected to each of its supply ports, including any implicit power and ground ports.

4.4.2.6 Supply set required functions

Although a supply set represents a collection of six standard supply functions, not all functions are required in every context:

- power and ground are typically required in all cases.
- nwell, pwell, deepnwell, and deeppwell are only required occasionally.

The required functions of a given supply set are determined from its usage and include the following:

- a) Any function used to define a power state of the supply set,
- b) Any function used for automatic connection of the supply set based on *pg_type*, and
- c) Any required function of a supply set handle with which the supply set is associated.

For implementation, a supply net shall be associated with each required function of a supply set. For verification, however, some aspects of the power intent can be verified before associating supply nets with the required functions. A supply set that does not have supply nets associated with each of its required functions is incompletely specified. For any required function of a supply set that is not associated with a supply net, an implicit supply net is created and associated with the function.

4.4.3 Supply equivalence

Various aspects of power management are determined in part by the identify of, and relationships between, supply nets and supply sets. For example, selection of ports to which isolation or level-shifting strategies should be applied can be defined based on the identities of the driver and receiver supplies of the sources and sinks connected to a port. Similarly, composition of power domains is possible provided the supplies of the subdomains involved meet certain constraints. In some situations, identical supply nets or supply sets are required; other situations may only require supply nets or supply sets that are equivalent.

There are two kinds of supply equivalence: electrical equivalence and functional equivalence.

Electrical equivalence can affect

- a) the number of sources of a supply network, and therefore,
- b) whether resolution is required for that supply network.

Electrical equivalence implies functional equivalence, but not vice versa.

Functional equivalence can affect any of the following:

- c) Insertion of isolation cells, level-shifter cells, and repeater cells
- d) Determination of power-domain lower boundaries
- e) Legality of power-domain composition
- f) Validity of driver and receiver supply attributes

Electrical equivalence is primarily related to supply ports and nets. Functional equivalence is primarily related to supply sets.

4.4.3.1 Supply port/net equivalence

Electrical equivalence is determined by connection, as follows:

- a) A port P is electrically equivalent to itself.
- b) A net N is electrically equivalent to itself.
- c) If a net N and a port P are connected, then N and P are electrically equivalent.
- d) If A and B are electrically equivalent, and B and C are electrically equivalent, then A and C are electrically equivalent.
- e) If A and B are connected via a supply set function (see 4.4.2.4), then A and B are electrically equivalent.

Electrical equivalence can also be declared, as follows:

— If A and B are declared electrically equivalent, then A and B are electrically equivalent.

Electrical equivalence implies the two equivalent objects are electrically connected somewhere. If the connection is not evident in the design (e.g., if it is inside a hard macro whose internals are not visible or if it is a connection that is required outside the design), then declaration of electrical equivalence can be used instead of the explicit connection.

Functional equivalence is determined by connection or declaration, as follows:

- f) If A and B are electrically equivalent, then A and B are functionally equivalent.
- g) If A and B are declared functionally equivalent, then A and B are functionally equivalent.

An input and the output of a switch are never electrically equivalent; it is an error if they are directly connected or declared electrically equivalent. Similarly, the outputs of two different switches are typically not electrically equivalent, unless they are both driving the same resolved net. However, the outputs of two different switches that each drive an unresolved net can still be functionally equivalent if the input supplies of both switches are equivalent, the control inputs of both switches are equivalent, and the two switches have the same set of state definitions.

4.4.3.2 Supply set equivalence

A supply set handle is also a supply set.

A supply set function and its associated supply net are electrically equivalent; thus, for purposes of supply net equivalence, a supply set function acts like a supply net.

Corresponding functions of two supply sets are electrically equivalent if

- their associated supply nets are electrically equivalent, or
- the two supply sets are directly associated with one another.

Corresponding functions of two supply sets are functionally equivalent if

- they are electrically equivalent, or
- they have been declared as functionally equivalent.

Two supply sets are (functionally) equivalent if

- they both have the same required functions, and the nets associated with corresponding functions are equivalent; or
- they are associated with each other directly or indirectly via one or more associate_supply_set commands (see <u>6.7</u>); or
- they are each associated directly or indirectly via associate_supply_set (see <u>6.7</u>) with two other supply sets, which are equivalent.

Two supply sets are also (functionally) equivalent if they have been declared equivalent; in this case, it is an error if they do not have the same required functions.

As a consequence of this,

- a) two anonymous supply sets built from equivalent PG functions are equivalent;
- b) two supply sets that are functionally equivalent can be used interchangeably;
- c) a supply set and any supply set handle it is associated with are always equivalent.

4.5 Power management

While a power supply network is a static structure, the power delivered via the power supply network can vary over time. Supply sources can provide different voltages; power switches can turn their outputs off or on and can selectively connect different inputs to the output. As a result, the power available to instances in the extent of a power domain will vary, and at any given time, each power domain's supplies may be in one of many possible states. To manage these various states, and in particular to manage the interactions between power domains that are in different states, power management is required.

Power management enables a system to operate correctly in a given functional mode with the minimum power consumption. Adding power management to a design involves analyzing the design to determine which power supplies provide power to each logic element, and if the driver and receiver are in different power domains, inserting power-management cells as required to ensure that neither logical nor electrical problems result if the two power domains are in different power states.

4.5.1 Related supplies

An active component consists of logic elements that receive inputs and drive outputs. The power supplies connected to an active component provide power for this logic. The supply nets that provide power for the logic that receives or drives a given input or output, respectively, are called the *related supplies* of that input or output. Related supplies typically include power and ground supplies and may also include bias supplies.

At the library cell level, related supplies may be identified for each input or output pin of a cell. Each related supply is a supply pin on that cell; the pin typically has a pg_type attribute indicating what supply function it provides (primary power, primary ground, etc.). For a cell that has one set of supply connections, all inputs and outputs would have the same set of related supplies. For a cell that has multiple supply connections, such as a cell with a backup power supply, different pins may have different sets of related supplies. This is particularly true of certain power-management cells, such as a level-shifter, which usually has different related supplies for the input and output.

Related supply nets are often considered in a group, as an implicit supply set. An implicit supply set made up of the supply pins of a cell that are the related supplies of a given input or output is by definition equivalent to any supply set that has been connected to those supply pins.

4.5.2 Driver and receiver supplies

Each output of an active component is typically connected to the input of some other active component in the design. The net connecting the two has a driver on one end (the logic driving the output port) and a receiver on the other end (the logic receiving the input). The driving logic is powered by a supply set called the *driver supply*; the receiving logic is powered by a supply set called the *receiver supply*.

The driver supply and the receiver supply may be the same supply set, e.g., if both components are in the same power domain; or the driver supply and the receiver supply may be different supply sets, e.g., if the two components are in different power domains. The driver supply and the receiver supply may also be

different, but nonetheless equivalent, e.g., if they are connected externally or if they are generated by supply networks that ensure they always have the same values.

In some cases, the logic driving or receiving a given port is not evident. In particular, the logic inside a hard macro instance may not be represented in a way that can be used by a given tool. Similarly, the logic that drives primary inputs of the design and receives primary outputs of the design is typically not represented as part of the design. In such cases, it is convenient to be able to associate the driver supply or receiver supply of the missing logic with the port that is connected to that logic. UPF defines attributes that can be used to associate this information with ports of a model.

4.5.3 Logic sources and sinks

Logic ports can be a source, a sink, or both, as follows:

- The LowConn of an input or inout logic port whose HighConn is connected to an external driver is a source.
- The HighConn of an output or inout logic port whose LowConn is connected to an internal driver is a source.
- The LowConn of an output or inout logic port whose Highconn is connected to an external receiver is a sink.
- The Highconn of an input or inout logic port whose LowConn is connected to an internal receiver is a sink.

For a logic port that is connected to a driver, the supply of the connected driver is also the driver supply of the port. A primary input port is assumed to have an external driver and therefore is a source; such a port has a default driver supply if it does not have an explicitly defined **UPF_driver_supply** attribute. An internal port that is not connected to a driver is not a source, and therefore, does not have a driver supply in the design. To model this in verification, an anonymous default driver is created for such an undriven port. This driver always drives the otherwise undriven port in a manner that results in a corrupted value on the port.

For a logic port that is connected to one or more receivers, the supplies of the connected receivers are all receiver supplies of the port. A primary output port is assumed to have an external receiver and therefore is a sink; such a port has a default receiver supply if it does not have an explicitly defined **UPF_receiver_supply** attribute. An internal port that is not connected to a receiver is not a sink, and therefore, does not have any receiver supplies.

4.5.4 Power-management requirements

Power management is required to mediate the changing power states of power domains in the system and the interactions between power domains that are in different states at various times. There are four specific areas addressed by power management, as follows:

- If a power domain is powered down in certain situations, its state registers may need to have their values saved before power-down and restored after subsequent power-up, either to maintain persistent data or to enable faster power-up.
- If the distance between driver and receiver is long (the capacitive load is high), buffers (repeaters) may be required to strengthen the signal along the way, or to ensure that it stabilizes within the required time.
- If a receiver is powered on, but its driver is not, an isolation cell is required between driver and receiver to drive the receiver with a known value despite the fact that the ultimate driver is powered off.
- If the driver and receiver supplies (or isolation and receiver supplies, or driver and isolation supplies, etc.) are operating at different voltage levels, a level-shifter is required between them to translate between voltage levels.

UPF provides commands for specifying where power-management structures should be added to a design to address each of these areas.

4.5.5 Power-management strategies

Addition of power-management cells to a design is driven by rules or strategies. UPF provides commands for specifying retention strategies (see 6.49), repeater strategies (see 6.48), isolation strategies (see 6.41), and level-shifting strategies (see 6.43). Each of these strategies can be defined in various ways to apply to specific design features or more generally to classes of features. Precedence rules (see 5.8) define how multiple strategies for the same feature are to be interpreted. In general, more specific strategies take precedence over more general strategies.

Retention strategies apply to specific state variables in a given power domain or to all state variables in a domain. A retention strategy also defines the power supplies, the control signals and their interpretation, and certain behavioral characteristics of the retention registers to be used for the state variables to which it applies.

Repeater, isolation, and level-shifting strategies apply to ports of a power domain. The ports to which one of these strategies applies can be defined by name or can be selected by filters. Source and sink filters select ports based on the driver supply and receiver supply, respectively, of each port. The filters typically match equivalent supplies unless an exact match is specified. Ports may also be selected by direction. Each of these strategies also specifies the relevant power supplies and control signals and their interpretation to be used for any power-management cells added by the strategy.

4.5.6 Power-management implementation

Implementation of power-management strategies involves adding power-management cells—retention registers, repeaters (buffers), isolation cells, and level-shifter cells—to the design. Each added cell may add new driving and receiving logic and as a result may change the driver and receiver supplies of a given port, which could potentially affect the application of other strategies based on source and sink filters. To ensure the interaction of multiple strategies is well defined, strategies are applied according to the following rules.

- a) Strategies are implemented in the following order: retention strategies, followed by repeater strategies, followed by isolation strategies, followed by level-shifter strategies.
- b) A retention strategy may affect the driving supply of the retention cell output. If so, the new driving supply of the retention cell is visible to, and affects the result of, a source filter of any subsequently applied strategy.
- c) A repeater strategy causes insertion of a buffer, which has a receiver and a driver; this insertion therefore affects both the receiving supply of ports driving the repeater input and the driving supply of ports receiver supply are visible to, and affect the result of, source and sink filters, respectively, of any subsequently applied strategy.
- d) An isolation strategy may cause insertion of an isolation cell, which has a receiver and a driver; therefore if such insertion occurs, it affects both the receiving supply of ports driving the isolation cell input and the driving supply of ports receiving the isolation cell output. However, the new driving supply and receiver supply are not visible to, and do not affect the result of, source and sink filters, respectively, of any subsequently applied isolation or level-shifting strategies.
- e) A level-shifting strategy may cause insertion of a level-shifting cell, which has a receiver and a driver; therefore if such insertion occurs, it affects both the receiving supply of ports driving the level-shifting cell input and the driving supply of ports receiving the level-shifting cell output. However, the new driving supply and receiver supply are not visible to, and do not affect the result of, source and sink filters, respectively, of any subsequently applied level-shifting strategy.

Repeater, isolation, and level-shifting strategies apply to all ports on the interface of a power domain, both those on the upper boundary of the domain and those on the lower boundary of a domain. As a result, a port on the boundary between two domains—the upper boundary of one, and the lower boundary of the other—may have multiple strategies of a given type defined for it, one from each of the two domains. In such a case, both strategies may cause addition of power-management cells.

4.5.7 Power control logic

Power-management elements require control signals to coordinate their activity. In particular, isolation cells require enable signals, retention cells may require save and restore signals or related control inputs, and power switches (see 4.4.1.2) require switch control signals. Logic ports and nets that implement these control signals may be present already in the HDL design or they may be added via UPF commands.

Control logic ports and nets defined in UPF are created within the logic hierarchy independent of powerdomain definitions. This allows the power control network to be created and distributed across power domains.

4.6 Power states

Supply ports, supply nets, supply sets, and power domains have associated power states. The power state of a supply port or net at a given time is the value propagated by that port or net. For a supply set or power domain, power states are defined based on supply port/net power states and other conditions.

Power switches also have named states. These are not power states of the switch, but rather states of the control expressions that determine which inputs of a switch affect the switch output (see 4.4.1.2).

4.6.1 Power state of a supply port or supply net

Supply ports and nets are represented by type **supply_net_type**, defined in package UPF (see <u>Annex B</u>). This type models electrical values as a combination of two values: a supply state and a voltage level, which together constitute the power state of the supply port or net.

The supply state value may be **OFF**, **UNDETERMINED**, **PARTIAL_ON**, or **FULL_ON**. The supply state value is not affected by or determined by the supply voltage level.

The voltage level is represented as an integer number of microvolts. The voltage level is relevant only for the **PARTIAL_ON** and **FULL_ON** supply states; it is undefined for the **OFF** and **UNDETERMINED** supply states.

4.6.2 Power state of a supply set

A supply set consists of a collection of functions that represent supply nets. A supply set has a reference supply net. The default reference is an implicit supply net with a supply state of **FULL_ON** and a voltage value of zero. The default reference supply can be explicitly overridden by specifying a supply net that is used as the reference supply for every supply net in the set. The voltage value of each supply net in a supply set is relative to the reference supply, which, in turn, may be at any voltage relative to the implicit reference supply.

Power states of a supply set are defined in terms of the power states of the supply functions that comprise the supply set, and the supply nets those functions represent, as well as related control conditions. The combined states of the constituent supply functions/nets and control conditions determine the following:

- Whether there is current available to power an instance, and
- The voltage level of the supply.

Power state definitions for a supply set are predicates: each one defines a set of conditions that, if satisfied, indicates that the supply set is in the corresponding state. Power state definitions need not be mutually exclusive; multiple power state definitions can be satisfied at any given time.

A supply set handle is also a supply set. Power states may be defined for a supply set handle as well as for a supply set.

Power state definitions for supply sets and supply set handles are only associated with and only apply to the supply set or supply set handle for which they are explicitly defined. They do not propagate to or apply to other associated supply sets or supply set handles.

4.6.3 Predefined supply set power states

Every supply set has two predefined power states: **DEFAULT_NORMAL** and **DEFAULT_CORRUPT**. These power states are identical to explicitly defined power states except: It is an error if **DEFAULT_NORMAL** and **DEFAULT_CORRUPT** are used as the *state_name* in an **add_power_state** command (see <u>6.4</u>).

A supply set is in the **DEFAULT_NORMAL** state when all of its required supply functions are **FULL_ON**.

The supply set is in the **DEFAULT_CORRUPT** power state when it is not in one of the defined power states of the supply set, including the **DEFAULT_NORMAL** predefined state, for the supply set.

4.6.4 Power states of power domains

A power domain typically represents a collection of instances that are powered with the same supplies. Power states of such a domain can be defined in terms of the power states of supply sets associated with the domain. Such definitions implicitly act as constraints on the power supplies provided to the domain.

A power domain may also be used to represent the interface to an IP block, which may contain multiple power domains. Power states of such a domain can be defined in terms of power states of the other domains in the IP block. Such definitions typically represent abstract power states of any given instance of the IP block.

For example, the definition of a domain's POWER_ON power state would logically require the primary supply set be in a power state in which all supply nets of the primary supply set are on and the current delivered by the power circuit is sufficient to support normal operation. Similarly, a SLEEP power state for the domain may require the primary supply set to be in power state in which sufficient voltage and current is provided to maintain the state of registers, but not enough to support normal operation. A POWER_OFF power state may require the primary supply set to be switched off, while the appropriate retention and isolation supplies are on.

The state of logic elements may be a relevant aspect to the specification of a domain's power state, e.g., for a user-defined power domain called DSP_PD,

- a) DSP_PD is in the state my_on_pd_state when:
 - 1) The logic signal that controls the switch for the domain's primary supply set is active.
 - 2) The logic signal(s) enabling isolation are inactive.
- b) DSP_PD is in the state my_off_pd_state when:
 - 1) The logic signal that controls the switch for the primary supply is inactive.
 - 2) If the isolation or retention supplies are switched, the control signals for those supplies are active (the power switch is on).
 - 3) Clock gating enable signals for the domain are typically inactive.

- 4) The isolation enable(s) are active.
- 5) The retention control signal(s) are active.
- c) The power domain's power state may also be dependent on the clock period or similar signal interval constraint. For example, a domain in an operational bias mode needs to scale its clock frequency to a slower level to match the slower switching performance supported by the state of the primary supply set. The primary supply set's power state can include in the **-logic_expr** specification a constraint on the clock period or duty cycle interval. See <u>6.4</u>.

4.6.5 Power states of systems and subsystems

What constitutes a system is contextual. In one context, a system may be considered as complete by itself, e.g., one chip of a multi-chip or multi-board low-power system. Although it might seem reasonable to define a "system" as that which is automatically implemented, UPF is not limited to that context and the verification of an entire system composed of multiple chips each with its own power intent specification, as well as an overall power intent specification for the board on which the chips are placed, is supported. The power states of a system or subsystem are attributed on a power domain. The use of the term *system* includes the term *subsystem*.

As a system power state may depend on the state of more than one power domain, the power state specification for a power domain may include references to the states of domains defined on scopes in the logic hierarchy that are descendants of the "higher-level" power domain. (Here, "higher-level" refers to the location of the power domain's scope being closer to the design's top-level root instance relative to the scope of another power domain.) Therefore, UPF allows a power state definition of a given power domain to reference the power state of any power domain or supply set, or the port state of any supply port or supply net, that is declared in the descendant subtree of the scope of that power domain.

For example, assume the domain CORE_PD is defined on the root scope of a processor design, the power states of CORE_PD can reference lower-level power domains such as CACHE_PD, ALU_PD, and FP_PD in the specification of its power states. Thus, an example power state of FULL_OP for CORE_PD would reasonably require that its primary supply set is in a **NORMAL** simstate (all supply nets of the primary supply set are on and the voltage of the supply is sufficient for normal operations) and that the CACHE_PD, ALU_PD, and FP_PD are all in an equivalent fully operational mode. In contrast, a NON_FP_OP mode for the CORE_PD may be defined identically to FULL_OP, except the FP_PD may be in a SLEEP mode. By allowing a higher level domain to reference lower-level domain's power states in the specification of its own power states, subsystem and system power states can be defined in UPF.

NOTE—Although the top-down specification of power states suggests a power domain's power states are defined in terms of the power states of supply sets and lower-level power domains, the power state of a domain can be specified entirely in terms of the state of supply sets and/or supply nets and supply ports; i.e., the hierarchical specification can be collapsed into a (relatively) flat power state specification. Top-down, hierarchical power state specification is convenient when the power design starts prior to the existence of the complete supply network and is refined into an implementation. The flat specification of power states of domains in terms of direct references to supply nets may be faster and more concise when the power state specification is not captured until after the supply network is specified. However, flat power state specifications may be less flexible and more difficult to maintain over time and require visibility into and understanding of all aspects of the design.

4.6.6 Incremental refinement of power states

Prior to having the golden source (the HDL and UPF source used as input to implementation tools), the supply network may not be defined or may be partially defined. The design may have a power-management block and associated power control signals that turn power switches on/off or control bias generators and voltage regulators once the supply network is fully specified. At this stage of design specification, the power domain's power states may be defined only in terms of the state of logic elements, i.e., control signals.

Power states of the domain's supply set handles may be added later as the supply network definition is completed. Through support of incremental refinement of the power state specification, early UPF simulations can be performed with only the logic net expressions defining the states. The power state definitions can be updated with **add_power_state** (see <u>6.4</u>) to incorporate supply network expressions (-supply expr) or additional logic expressions (-logic expr).

4.7 Simstates

Simstates specify the simulation behavior semantics for a power state. A simstate specifies the level of operational capability supported by a supply set state. The simstate specification provides digital-simulation tools sufficient information for approximating the power-related behavior of logic connected to the supply set with sufficient accuracy.

Simstates are associated with power states of supply sets and supply set handles. A simstate defines how instances powered by the supply set or supply set handle react to a given power state. In particular, simstates can be associated with power states of the primary supply of a power domain, to define how instances in the power domain that are implicitly connected to that primary supply will behave under various power states of the primary supply.

UPF defines several simstates that can be associated with supply set or supply set handle power states. The simstates defined in UPF are an abstraction suitable for digital simulation. The following simstates are defined (from highest to lowest precedence):

- a) **CORRUPT**—The supply set is either off (one or more supply nets in the set are switched off, terminating the flow of current) or at such a low-voltage level that it cannot support switching and the retention of the state of logic nets cannot be guaranteed to be maintained even in the absence of activity in the instances powered by the supply.
- b) **CORRUPT_ON_ACTIVITY**—The power characteristics of the supply set are sufficient for logic nets to retain their state as long as there is no activity within the elements connected to the supply, but they are insufficient to support activity.
- c) **CORRUPT_ON_CHANGE**—The power characteristics of the supply set are sufficient for logic nets to retain their state as long as there is no change in the outputs of the elements connected to that supply.
- d) **CORRUPT_STATE_ON_ACTIVITY**—The power characteristics of the supply set are sufficient to support normal operation of combinational logic, but they are insufficient to support activity inside state elements, whether that activity would result in any state change or not.
- e) **CORRUPT_STATE_ON_CHANGE**—The power characteristics of the supply set are sufficient to support normal operation of combinational logic, and they are sufficient to support activity inside state elements, but they are insufficient to support a change of state for state elements.
- f) **NORMAL**—The power characteristics of the supply set are sufficient to support full and complete operational (switching) capabilities with characterized timing.

The predefined power states for a supply set have corresponding simstates. The simstate for power state **DEFAULT_NORMAL** is **NORMAL**. The simstate for power state **DEFAULT_CORRUPT** is **CORRUPT**.

Simstate simulation semantics for a supply set are applied to instances implicitly connected to a supply set unless simstate behavior has been disabled (see 6.53).

NOTE 1—When greater accuracy is desired or required, a mixed signal or full analog simulation can be used. Since analog simulations already incorporate power, this format provides no additional semantics for analog verification.

Simulation results reflect the implemented hardware results only to the extent the UPF simstate specification for a given power state of a supply set is correctly specified. For example, if verification is performed with simulation of a supply set in a power state specified as having a **CORRUPT_ON_ACTIVITY** simstate, but the implementation is more accurately classified as **CORRUPT_STATE_ON_CHANGE**, the simulation results will differ.

NOTE 2—In this example, the inaccuracy in simstate specification is conservative relative to the implemented hardware behavior. However, in other situations, inaccurate specifications can be optimistic, resulting in errors in the implemented hardware that simulation failed to expose.

4.8 Successive refinement

Design and implementation of a power-managed system using UPF proceeds in stages. During the design phase, a UPF-based specification of the power intent may be developed incrementally, first at the IP block level, and later at the system level. During implementation, UPF commands are added to drive implementation details, and a series of implementation steps map the design and the UPF commands into the final implementation (see Figure 2).



Figure 2—Successive refinement of power intent

The power intent specification for an IP block to be used in a larger design typically defines the power interface to the block and the power domains within the block. This specification also typically includes constraints on the use of the block in a power-managed environment. These constraints include (at least) the following:

a) The atomic power domains in the design.

These can be composed but not split during implementation. [Use **create_power_domain – atomic** (see <u>6.17</u>).]

- b) The state variables that need to have their values retained if a given power domain is powered down. This does not involve specifying how such retention would be controlled. [Use set_retention_elements (see 6.51).]
- c) The clamp values of signals that would need to be isolated if a given power domain is powered down.

This does not involve specifying how isolation is to be controlled. [Use set_port_attributes $-clamp_value$ (see <u>6.46</u>).]

d) The legal power states and power state transitions of the IP block's power domains.

This need not involve specifying absolute voltage ranges for the power supplies involved. [Use add_power_state (see 6.4) and describe_state_transition (see 6.24).]

A power intent specification containing such basic information about an IP block is often referred to as *constraint UPF*, or sometimes as the *platinum UPF*.

When an IP block is being prepared for use in a given system, information may be added to the specification to reflect the specific requirements of the block in the context of the system. For example, an instance of the block may be used in a manner that will definitely require isolation, level-shifting, retention, or repeater cell insertion. These strategies can be added to the constraint UPF for the block in order to configure the power intent of the block for use in this system. Such strategies impose a requirement to insert specified power-management cells for an instance of the IP block and typically include information about how such power-management cells are controlled.

A power intent specification containing this level of information is often referred to as *configuration UPF*, or sometimes as the *golden UPF*.

To drive implementation of a power-managed design, information may be added to the specification to define the power distribution network for the system and the control logic for power-management cells. A power intent specification containing this kind of information is often referred to as *implementation UPF*, or sometimes as the *silicon UPF*.

4.9 Tool flow

A UPF-based tool flow typically begins with RTL verification of the design together with the golden UPF that defines the power intent for this design. After that, a series of implementation steps occur in which the RTL design is reduced to a gate-level implementation and the power intent is integrated into that implementation. After each implementation step, power-aware verification may be performed again, using the design representation output by that stage along with the UPF description corresponding to that design representation (see Figure 2).

The power intent expressed in UPF can be implemented incrementally in successive steps. Each step may add implementation details, such as power-management cells, control logic, or supply distribution networks. The design itself may also evolve during implementation, even after the RTL stage, as a result of implementation steps such as test insertion.

Implementation may be incremental at various levels of granularity as follows:

- By aspect: isolation, level-shifting, retention, repeaters, control logic, power distribution
- By command: isolation strategy A, isolation strategy B, etc.
- By element to which a command applies: isolation for port p1, for port p2, etc.

For any given tool run, the tool needs to know the following:

- a) What part of the UPF power intent specification is supposed to be implemented already, and
- b) What part of the UPF power intent specification is to be included in the processing done by this tool.

This standard does not define how the preceding information is made available to a tool; this is tool/flow information that is outside the scope of the standard. Typically, such information would be provided to the tool either explicitly via command-line arguments or other control inputs, or implicitly as part of the specification of the tool itself.

A tool also shall be able to determine what part of the UPF specification has been implemented so far. This standard defines a method for documenting what has been done so far to implement the power intent, by identifying ports, nets, and instances in the design that represent implementations of UPF commands.

4.10 File structure

For maximum reuse, it may be appropriate to keep constraint, configuration, and implementation UPF commands in separate files. The **load_upf** command (see 6.28) can be used to compose the files for a particular context.

For example, an IP block with a corresponding constraint UPF description might be configured for use in a given system by creating a configuration UPF file for it. The configuration UPF file would load the constraint UPF for the IP block and then continue with additional commands defining or updating the isolation, level-shifting, retention, and repeater strategies required for this configuration of the IP block. Different configuration UPF files can be constructed based on the same constraint UPF, to define different configurations of the same IP block for use in different situations.

For implementation of the design, an implementation UPF file may be constructed by loading the configuration UPF for the various IP blocks involved in the system and then adding implementation details, such as supply ports, nets, and sets, power switches, port attributes, and supply connections. Different implementation UPF files can be constructed using the same configuration UPF files, to evaluate or verify alternative implementations.

For each implementation step, tools may update the implementation UPF to document the additions made to the design in that step to implement the power intent. To keep the implementation updates separate from the input UPF specification, a tool may generate an output UPF file that loads the input UPF file and then adds UPF command updates as required. Successive implementation steps may choose to append to this update file or generate a new update file that loads the previous one.

5. Language basics

5.1 UPF is Tcl

UPF is based on Tool Command Language (Tcl). UPF commands are defined using syntax that is consistent with Tcl, such that a standard Tcl interpreter can be used to read and process UPF commands.

Compliant processors reading UPF files use full Tcl interpreters to process the UPF files. Compliant processors shall use Tcl version 8.4 or above. The following also apply:

- UPF power intent commands are executed in the order of occurrence, just as Tcl commands are
 executed and return values can be used by subsequent commands.
- The only UPF commands that support regular expressions are find_objects (see <u>6.26</u>) and query_upf (see <u>C.1</u>).
- All of the commands and techniques of Tcl may be used, including procs and libraries of procs. However, the procs and libraries of procs should, in the end, only rely on UPF commands for design information.
- find_objects (see <u>6.26</u>) shall be the only source used to programmatically access the HDL when defining the power intent. The processing of information returned by find_objects using standard Tcl commands [B5], such as regexp, is allowed.
- UPF is intended to be used across many tools, so it is erroneous to use proprietary tool specific commands when constructing power intent.
- Once the Tcl processing has completed, the end result can be expressed as a series of UPF commands.

Libraries used for design or methodology standardization or ease of expression that define additional procs are considered to be part of the design file and need to be visible to any processor interpreting the UPF file.

5.2 Conventions used

Each UPF command in <u>Clause 6</u> and <u>Clause 7</u> consists of a command keyword followed by one or more parameters. All parameters begin with a hyphen (-). The meta-syntax for the description of the syntax rules uses the conventions shown in <u>Table 1</u>.

Visual cue	Represents
courier	The courier font indicates UPF or HDL code. For example, the following line indicates UPF code:
	create_power_domain PD1
bold	The bold font is used to indicate key terms, text that shall be typed exactly as it appears. For example, in the following command, the keywords "create_power_domain" shall be typed as it appears: create_power_domain domain_name
italic	The <i>italic</i> font represents user-defined UPF variables. For example, a supply net shall be specified in the following line (after the "connect_supply_net" key term): connect_supply_net net_name

Table 1—Document conventions

Visual cue	Represents
list	<i>list</i> (or <i>xyz_list</i>) indicates a Tcl list, which is denoted with curly braces $\{\}$ or as a double-quoted string of elements "". When a list contains only one non-list element (without special characters), the curly braces can be omitted, e.g., $\{a\}$, "a", and a are acceptable values for a single element. See also <u>5.3.4</u> .
xyz_ref	<i>xyz_ref</i> can be used when a symbolic name (i.e., using a handle) is allowed as well as a declared name, e.g., <i>supply_set_ref</i> .
time_literal	time_literal indicates a SystemVerilog or VHDL time_literal.
* asterisk	An asterisk (*) signifies that a parameter can be repeated. For example, the following line means multiple acknowledge delays can be specified for this command: [-ack_delay {port_name delay}]*
[] square brackets	Square brackets indicate optional parameters. If an asterisk (*) follows the closing bracket, the bracketed parameter may be repeated. For example, the following parameter is optional:
	[-elements element_list]
	The following is an example of optional parameter that can be repeated: [-ack_port {port_name net_name [{boolean_expression}]}]*
[] bold square brackets	Bold square brackets are required. For example, in the following parameter, the bold square brackets (surrounding the 0) need to be typed as they appear: domain_name.isolation_name.isolation_supply_set[0]
{ } curly braces	Curly braces ({ }) indicate a parameter list that is required. In some (or even many) cases, they have (or are followed by) an asterisk (*), which indicates that they can be repeated. For example, the following shows one or more control ports can be specified for this command: {-control_port {port_name}}*
<pre>{} bold curly braces</pre>	Bold curly braces are required, unless the argument is already a Tcl list. For example, in the following parameter, the bold curly braces need to be typed as they appear: [-off_state {state_name {boolean_expression}}]*
	In cases where variable substitution is needed, Tcl's list command can be used, e.g., -off_state [list \$state_name [list \$expression]]
<> angle brackets	Angle brackets (< >) indicate a grouping, usually of alternative parameters. For example, the following line shows the "power" or "ground" key terms are possible values for the "-type" parameter: -type <power ground="" =""></power>
separator bar	The separator bar () character indicates alternative choices. For example, the following line shows the "in" or "out" key terms are possible values for the "-direction" parameter:
	-direction <in out="" =""></in>

Table 1—Document conventions (continued)

5.3 Lexical elements

Names created in UPF should not conflict with HDL reserved words.

Command names, parameter names, and their values are case-sensitive.

5.3.1 Identifiers

Identifiers adhere to the following rules:

- a) The first character of a identifier shall be alphabetic.
- b) All other characters of a identifier shall be alphanumeric or the underscore character ().
- c) Identifiers in UPF are case-sensitive.

5.3.2 Keywords and reserved words

The following record field names are reserved in the specified context and cannot be redefined:

- a) Domain record field space
 - 1) primary
 - 2) default_retention
 - 3) default_isolation
- b) Switch record field space
 - 1) supply
- c) Level-shifter strategy record field name space
 - 1) input_supply_set
 - 2) output_supply_set
 - 3) internal_supply_set
- d) Isolation strategy record field name space
 - 1) isolation_supply_set
 - 2) isolation_signal
- e) Retention strategy record field name space (see 6.33)
 - 1) retention_ref
 - 2) retention_supply_set
 - 3) primary ref
 - 4) primary_supply_set
 - 5) save_signal
 - 6) restore_signal
 - 7) UPF_GENERIC_CLOCK
 - 8) UPF_GENERIC_DATA
 - 9) UPF GENERIC ASYNC LOAD
 - 10) UPF_GENERIC_OUTPUT

5.3.3 Names

Names identify objects in the design and in the power intent specification.

5.3.3.1 Simple names

A simple name is a single identifier. An identifier is used when creating a new object in a given scope; the identifier becomes the simple name of that object.

In a given scope, a given simple name may only be defined once, with a unique meaning; it is an error if two objects are declared in the same scope with the same simple name.

A simple name, optionally followed by an index or record field specification as appropriate for the type of an object in a given HDL context, is an object name. An object name can be used to refer to an existing object or part of an existing object that is declared in the current scope. Object names also refer to objects defined in UPF that do not exist in a scope of the hierarchy.

The simple name of an instance in a given scope is an instance name.

5.3.3.2 Dotted names

A dotted name is a compound name designating a UPF object. A dotted name is made up of simple names separated by . characters.

A dotted name is used to refer to a strategy associated with a power domain, a supply set associated with a strategy or a power domain, or a function of a supply set. A dotted name for a supply set associated with a strategy or domain is called a *supply set handle*. A dotted name for a supply set function is called a *supply net handle*.

Power-domain strategy names

<domain name> . <strategy name>

Supply set handles

<domain name> . <supply set name>
 <domain name> . <strategy name> . <supply set name>
unduration diag

Supply net handles

<supply set name> . <function name> <domain name> . <supply set name> . <function name> <domain name> . <strategy name> . <supply set name> . <function name>

A dotted name is also an object name.

5.3.3.3 Hierarchical names

A hierarchical name is a name that refers to an object declared in a non-local scope. A hierarchical name consists of an optional leading / character, followed by a series of one or more instance names, each followed by the hierarchy separator character /, followed by an object name.

A hierarchical name that starts with an instance name is a scope-relative hierarchical name. A scope-relative hierarchical name is interpreted relative to the current scope. The first instance name is the name of an instance declared in the scope of the previous instance. The trailing object name is the simple name or dotted name of an object declared in the scope of the last instance. A scope-relative hierarchical name is also called a *rooted name*.

A hierarchical name that starts with a leading / character is a design-relative hierarchical name. A design-relative hierarchical name is interpreted relative to the current design top instance, by removing the leading / character and interpreting the remainder as a rooted name in the scope of the current design top instance.

5.3.3.4 Name references

Many command arguments require references to object names, such as the names of instances, ports, registers, nets, etc., in the design, or the names of power domains, strategies, supply sets, supply nets, etc., in the power intent. Unless otherwise specified or contextually restricted, an object name reference can be a simple name, a dotted name, or a hierarchical name. In particular, a supply set handle is a form of supply set name and a supply net handle is a form of supply net name. In the absence of any statement to the contrary, a supply set handle can be used wherever a supply set name may appear, and a supply net handle can be used wherever a supply net name.

5.3.4 Lists and strings

A Tcl list is an ordered sequence of zero or more elements, where each element can itself be a list. In Tcl, a string can be thought of as a list of words.

Tcl strings can be specified in two different ways: by enclosing the words within double-quotes ("") or between curly braces ({}). Upon finding a list of words within double-quotes, Tcl continues to parse the string, looking for variable (strings started with \$), command (strings between square brackets []), and back-slash (strings contain $\)$ substitutions. To use any of the special characters within design object names, first wrap them in curly braces ({}). Upon finding a list of words between curly braces, Tcl treats the list as a literal list of words, preventing further processing on the list before it is used.

Therefore, in the syntax for UPF, the construct **-option** xxx_list can be satisfied by any of the following, when no special characters are used in the object names:

```
-option foo
-option "foo"
-option "foo bar bat"
-option {foo}
-option {foo bar bat etc}
```

5.3.5 Special characters

Special lexical elements (see Table 2) can be used to delimit tokens in the syntax.

Туре	Character
Logic hierarchy delimiter	/
Escape character	(only escapes the next character)
Bus delimiter, index operator, or within a regex	П
Range separator (for bus ranges)	:
Record field delimiter	•

Table 2—Special characters

When Tcl special characters need to be used literally for design object names, always escape the special character or wrap the name with $\{\}$, even if a single value is used, to protect from Tcl interpretation, e.g., -elements [list foo {foo/bar} a\[0\]].

5.4 Boolean expressions

A Boolean expression may be used to define a control condition or a supply state. A Boolean expression may include references to the following.

- a) VHDL names, values, and literals of the following types or any subtype thereof:
 - std.Standard.Boolean
 - std.Standard.Bit
 - std.Standard.Real for voltage values
 - std.Standard.Time for use with the interval function

```
ieee.std_logic_1164.std_ulogic
ieee.UPF.state
```

b) SystemVerilog names, values, and literals of the following types:

```
reg
wire
Bit
Logic
time_literal for use with the interval function
real, shortreal for voltage values
```

A VHDL or SystemVerilog name may also be the name of an element of any composite type object provided the element itself is of a supported type.

A Boolean expression may also contain special expression forms for referring to power states (see <u>6.4</u>).

A name of an object referred to in a Boolean expression may be prefixed by a path name identifying the instance in the scope of which the name is declared. Any such path name is interpreted relative to the current scope when the command defining the expression is executed. If no path name prefix is present, the name shall refer to an object declared in the current scope.

In a Boolean expression used as a supply expression in the definition of a power state of a supply set (handle), the name of any function of that supply set (handle) may be referred to directly without a prefix, unless such a reference would be ambiguous.

In a Boolean expression used as a logic expression in the definition of a power state of a power domain, the name of any supply set handle associated with that power domain may be referred to directly without a prefix, unless such a reference would be ambiguous.

A Boolean expression may include the operators shown in <u>Table 3</u>, which map to their corresponding equivalents in SystemVerilog or VHDL, as appropriate for the objects involved in each subexpression.

Operator	SystemVerilog equivalent	VHDL equivalent	Meaning
!	!	not	Logical negation
~	~	not	Bit-wise negation
<	<	<	Less than
<=	<=	<=	Less than or equal
>	>	>	Greater than
>=	>=	>=	Greater than or equal
==	==	=	Equal
!=	! =	/=	Not equal
&	á	and	Bit-wise conjunction
٨	^	xor	Bit-wise exclusive disjunction

Table 3—Boolean	operators
-----------------	-----------

Operator	SystemVerilog equivalent	VHDL equivalent	Meaning
		or	Bit-wise disjunction
&&	& &	and	Logical conjunction
		or	Logical disjunction

Table 3—Boolean operators (continued)

A Boolean expression shall be provided as a string, as indicated in the syntax for each command in which a Boolean expression can appear. Subexpressions may be grouped with parenthesis (()). Logical operators have lowest precedence; bit-wise operators have next higher precedence; relational operators have next higher precedence; negation operators have highest precedence.

A Boolean expression or subexpression is considered to evaluate to the logical value *True* if evaluation of the expression (according to the semantics of the VHDL or SystemVerilog operators and types involved, as appropriate) results in a bit or logic value of 1 or a Boolean value of *True*; otherwise it is considered to evaluate to the logical value *False*.

A Boolean expression may contain references to objects in different language contexts provided that any given subexpression that evaluates to a logical (*True/False*) value contains only references to one language context. Logical negation, conjunction, and disjunction of logical values shall be performed according to standard Boolean logic semantics and need not be implemented with language-specific operators.

A simple expression is a Boolean expression containing an optional negation operator (! or \sim), followed by optional white space and a single object name.

Examples

```
{ top/sv_inst/ena == 1'b1 && top/vhdl_inst/ready == `0' }
{ supply1.state == FULL_ON && supply1.voltage > 0.8 }
{(top/sv/wall.supply[0] != FULL_ON) || (top/vhdl/battery.supply(1) ==
    UNDEFINED) }
```

5.5 Object declaration

All UPF commands are executed in the current scope, except as specifically noted.

As a result, most objects created by a UPF command are created in the current scope within the design; therefore, the names of those objects shall not conflict with a name that is already declared within the same scope.

Some UPF objects are implicitly created. *Implicitly created objects* result from implied or inferred semantics and are not the direct result of creating a named UPF object. For example, supply nets are routed throughout the extent of a power domain as needed to implement the implicit and automatic connection semantics. This routing results in the creation of implicit supply ports and supply nets. UPF automatically names implicitly created objects to avoid creating a name conflict. The **name_format** command (see <u>6.35</u>) can be used to provide a template for some implicitly created objects (such as isolation). Supply nets may be implicitly created and connected to logic ports (see <u>4.4.1.1</u>).

UPF objects may have record fields. These records comprise a name and a set of zero or more values. Record field names are in a local name space of the UPF object, e.g., a power domain may have strategies and supply set handles. Strategies themselves may also have supply set handles.

The . character is the delimiter for the hierarchy of UPF record fields, e.g., top/a/ PDa.MY_SUPPLY_SET refers to the supply set MY_SUPPLY_SET in power domain PDa in the logical scope top/a.

5.6 Attributes of objects

HDLs include a mechanism for specifying properties of objects. These properties are called *attributes*. Certain UPF properties can be annotated directly in HDL source descriptions using attributes. The semantic for properties specified using HDL attributes is the same as the corresponding behavior defined by the UPF command alternative (see <u>Clause 6</u>). <u>Table 4</u> enumerates the HDL attributes defined for UPF-compliant implementations.

HDL attribute name	Attribute value specification	Equivalent UPF command arguments	See
UPF_clamp_value	<"0" "1" "Z" "latch" "any" " <i>value</i> ">	set_port_attributes -clamp_value	<u>6.46</u>
UPF_sink_off_clamp_value	<"0" "1" "Z" "latch" "any" " <i>value</i> ">	set_port_attributes -sink_off_clamp_value	<u>6.46</u>
UPF_source_off_clamp_value	<"0" "1" "Z" "latch" "any" " <i>value</i> ">	set_port_attributes -source_off_clamp_value	<u>6.46</u>
UPF_pg_type	<i>pg_type_value</i> (see <u>4.4.2.5</u>)	set_port_attributes -pg_type	<u>6.46</u>
UPF_related_power_port	supply_port_name	set_port_attributes -related_power_port	<u>6.46</u>
UPF_related_ground_port	supply_port_name	set_port_attributes -related_ground_port	<u>6.46</u>
UPF_related_bias_ports	supply_port_name _list	set_port_attributes -related_bias_ports	<u>6.46</u>
UPF_driver_supply	supply_set_ref	set_port_attributes -driver_supply	<u>6.46</u>
UPF_receiver_supply	supply_set_ref	set_port_attributes -receiver_supply	<u>6.46</u>
UPF_feedthrough	<"TRUE" "FALSE">	set_port_attributes -feedthrough	<u>6.46</u>
UPF_unconnected	<"TRUE" "FALSE">	set_port_attributes -unconnected	<u>6.46</u>
UPF_retention	<"required" "optional">	set_retention_elements -retention_purpose	<u>6.51</u>
UPF_simstate_behavior	<"ENABLE" "DISABLE">	set_simstate_behavior	<u>6.53</u>

Table 4—Attribute and command correspondence

HDL attribute name	Attribute value specification	Equivalent UPF command arguments	See
UPF_is_leaf_cell	<"TRUE" "FALSE">	set_design_attributes -is_leaf_cell	<u>6.37</u>
UPF_is_macro_cell	<"TRUE" "FALSE">	set_design_attributes -is_macro_cell	<u>6.37</u>

Table 4—Attribute and command correspondence (continued)

The HDL attributes in <u>Table 4</u> all take values that are string literals. Where a list of names is required, the names in the list should be separated by spaces and without enclosing braces ({}). To attach a UPF attribute to an object in a VHDL context, the UPF attribute shall be declared first, with a data type of STD.Standard.String (or the equivalent), before any attribute specification for that attribute.

It shall be an error if any of the attributes in <u>Table 4</u> is defined multiple times with different values for the same object, regardless of whether the attribute is defined as an HDL attribute or using UPF commands or both.

Examples

A port-supply relationship can be annotated in HDL using the following attributes:

Attribute name: UPF_related_power_port and UPF_related_ground_port.

Attribute value: "*supply_port_name*", where *supply_port_name* is a string whose value is the simple name of a port on the same interface as the attributed port.

SystemVerilog or Verilog-2005 attribute specification:

(* UPF_related_power_port = "my_VDD", UPF_related_ground_port = "my_VSS" *) output my Logic Port;

VHDL attribute specification:

```
attribute UPF_related_power_port : STD.Standard.String;
attribute UPF_related_power_port of my_Logic_Port : signal is
"my_VDD";
attribute UPF_related_ground_port : STD.Standard.String;
attribute UPF_related_ground_port of my_Logic_Port : signal is
"my_VSS";
```

Attribute name: UPF_related_bias_pin.

Attribute value: "*supply_port_name_list*", where *supply_port_name_list* is a string whose value is a space-separated list of one or more simple names of port(s) on the same interface as the attributed port.

SystemVerilog or Verilog-2005 attribute specification:

```
(* UPF_related_bias_ports = "my_VNWELL my_VPWELL" *)
output my Logic Port;
```

VHDL attribute specification:

```
attribute UPF_related_bias_ports : STD.Standard.String;
attribute UPF_related_bias_ports of my_Logic_Port : signal
is "my VNWELL my VPWELL";
```

The same attributes can be specified in UPF, using the **set_port_attributes** command and its generic **-attribute** option, or they can also be specified in UPF using the **set_port_attributes** command and its specific options **-related_power_port**, **-related_ground_port**, and **-related_bias_ports**, respectively (see <u>6.46</u>).

Isolation clamp value port properties can be annotated in HDL using the following attributes:

Attribute name: UPF_clamp_value

Attribute value: <"0" | "1" | "Z" | "latch" | "any" | "value">

SystemVerilog or Verilog-2005 attribute specification:

(* UPF clamp value = "1" *) output my Logic Port;

VHDL attribute specification:

attribute UPF_clamp_value : STD.Standard.String;

attribute UPF clamp value of my Logic Port : signal is "1";

The same attributes can be specified in UPF, using the **set_port_attributes** command and its generic **-attribute** option, or it can also be specified in UPF, using the **set_port_attributes** command and its specific option **-clamp_value** (see <u>6.46</u>).

pg_type port properties can be annotated in HDL using the following attributes:

```
Attribute name: UPF_pg_type
```

```
Attribute value: <"primary_power" | "primary_ground" |
"backup_power" | "backup_ground" >
```

SystemVerilog or Verilog-2005 attribute specification:

(* UPF pg type = "primary power" *) output myVddPort;

VHDL attribute specification:

```
attribute UPF_pg_type : STD.Standard.String;
attribute UPF_pg_type of myVddPort : signal
    is "primary_power";
```

The same attributes can be specified in UPF, using the **set_port_attributes** command and its generic **-attribute** option, or it can also be specified in UPF using the **set_port_attributes** command and its specific option **-pg_type** (see <u>6.46</u>).

The UPF leaf-cell treatment of a model or instance can be annotated in HDL using the following attributes:

Attribute name: UPF_is_leaf_cell

Attribute value: <"TRUE" | "FALSE">

SystemVerilog or Verilog-2005 attribute specification:

(* UPF is leaf cell="TRUE" *) module FIFO (<port list>);

VHDL attribute specification:

attribute UPF_is_leaf_cell : STD.Standard.String; attribute UPF is leaf cell of FIFO : entity is "TRUE";

The same attribute can be specified in UPF, using the set_design_attributes command (see 6.37).

When any register (specified or implied) with the **UPF_retention** attribute value set to **"required"** is included in a power domain that has at least one retention strategy, the register shall be included in a retention strategy defined for the domain.

Elements requiring retention can be attributed in HDL as follows:

Attribute name: UPF_retention

Attribute value: <"required" | "optional">

SystemVerilog or Verilog-2005 attribute specification:

(* UPF_retention = "required" *) module my_mod;

VHDL attribute specification:

attribute UPF retention : STD.Standard.String;

attribute UPF retention of my flip : variable is "required";

The same attribute can be specified in UPF, using the **set_retention_elements** command and its specific option **-retention_purpose** (see <u>6.51</u>).

5.7 Power state name spaces

Power states are attributed to specific objects in the design. The power states can be referenced by specifying the *object_name*, where *object_name* can be a hierarchical name denoting a power domain, supply set, or supply net. Power states are attributes of the object. Specifically, *power states* of a domain are attributes of the domain and not attributes of the scope of the domain. Thus, an instance may be the scope for multiple domains, each domain containing states with the same name (e.g., sleep) without incurring a name space collision.

The following objects may have power states attributed to them:

- Power domains
- Supply sets
- Supply nets
- Supply ports

The **add_power_state** command (see <u>6.4</u>) is used to define the legal and illegal power states of power domains and supply sets. The set_power_state function in the package UPF is used to set the power state of an object during simulation.

The range of possible states for supply nets and ports is defined by the type <code>supply_net_type</code> in the package UPF. The state of supply nets and ports can be set through the <code>assign_supply2supply</code> or <code>assign_supply_state</code> functions in the package UPF. <code>assign_supply2supply</code> propagates the association of the source supply net's root supply driver as well as the source's state and voltage values to the destination. <code>assign_supply_state</code> is used to assign a supply port that is a root supply driver.

A power state shall be defined before it can be referenced. Semantically, the transition of an object from one power state to another is a power state *event* for the object. The state of a supply net is referenced as a Boolean expression (see 5.4) in the same manner that the state of a logic net is referenced. The power state of a supply set or power domain can be referenced in an expression simply through the supply set or power-domain name.

Examples

```
supply_set_li == SLEEP
-- Returns TRUE if supply_set_li is in a state consistent with state SLEEP
ALU_PD != FULL_OP
-- Returns TRUE if the ALU PD is in a state inconsistent with FULL OP
```

5.8 Precedence

To support concise, easily written low-power specifications, UPF commands may range from very specific to very generic in their scope of application. This enables specification of generic defaults that apply widely except where more specific commands provide more focused information. This subclause describes the precedence relations that determine which of several commands that potentially apply in a given situation will actually apply.

A **create_power_domain** command (see <u>6.17</u>) that explicitly includes a given instance in its extent shall take precedence over one that applies to an instance transitively (i.e., applies to an ancestor of the instance, and therefore to all of its descendants). A **create_power_domain** command that creates an atomic power domain takes precedence over one that creates a non-atomic power domain.

If multiple **set_isolation** commands (see 6.41), or multiple **set_level_shifter** commands (see 6.43), or multiple **set_repeater** commands (see 6.48) potentially apply to the same port, the following criteria (listed in order from highest precedence to lowest precedence) determine the relative precedence of the commands, and only the command(s) with the highest precedence will actually apply:

- a) Command that applies to part of a multi-bit port specified explicitly by name
- b) Command that applies to a whole port specified explicitly by name
- c) Command that applies to all ports of an instance specified explicitly by name
- d) Command that applies to all ports of a specified power domain with a given direction
- e) Command that applies to all ports of a specified power domain

If multiple strategies of the same type have the same highest precedence, then all of those commands actually apply to the port or part thereof, to the extent allowed by the strategy.

A prefix or suffix to be used to create names for inserted isolation, level-shifter, and repeater cells that is specified by the **-name_prefix** or **-name_suffix** options, respectively, of **set_isolation**, **set_level_shifter**, and **set_repeater**, takes precedence over any user-defined prefix or suffix for these commands specified by the **name_format** command (see <u>6.35</u>). A prefix or suffix explicitly specified using the **name_format** command in turn takes precedence over the default prefix or suffix specified in the definition of the **name_format** command.

If multiple supply connections potentially apply to the same port, the actual application is determined by the following precedence order, from highest to lowest precedence:

- f) Command that explicitly connects to part of a port
- g) Command that explicitly connects to a whole port (e.g., connect_supply_net -ports/-pins)
- h) Command that automatically connects to ports of an instance (
 e.g., connect_supply_set -connect -elements)
- Command that automatically connects to ports of any instance in a given region (e.g., connect_supply_set -connect or connect_supply_net -pg_type -domain/-cells)

Any explicit connection command takes precedence over implicit connections made by default.

For attribute specifications, there is no definition of precedence to select which of several potentially applicable specifications apply. It is an error if any two UPF, HDL, or Liberty attribute specifications provide different values for the same attribute of the same object.

For simstates that apply to a given object at any given time, a more conservative (i.e., more corrupting) simstate takes precedence over a less conservative (less corrupting) simstate.

The following also apply:

- j) The precedence of a command is independent of the current scope during the command processing.
- k) It shall be an error if the precedence rules fail to uniquely identify the power intent that applies to an object.
- The find_objects command (see <u>6.26</u>) returns a list of explicit names; these names can refer to whole objects or to elements thereof. When *list* arguments to command options are created using find_objects, the level of precedence is based on the expanded value used as the argument, not as the pattern or regular expression used in find_objects.
- m) The symbol . in -elements {.} is an explicit reference to the instance corresponding to the current scope.

5.9 Generic UPF command semantics

All **map_*** commands specify the elements to be used in implementation. These specifications override the elements that may be inferred through a strategy. The behavior of this manual mapping may lead to an implementation that is different from the RTL specification. Therefore, it may not be possible for logical equivalence checking tools to verify the equivalence of the mapped element to its RTL specification.

5.10 effective_element_list semantics

The *effective_element_list* is the set of elements to which a command applies. The *effective_element_list* is constructed from the arguments provided to the command. The terms used in the description of this construction include: *element_list, exclude_list, aggregate_element_list, aggregate_exclude_list, prefilter_element_list,* and *effective_element_list.* The *element_list* and *exclude_list* are lists that contain the elements specified by an instance of the command. The *effective_element_list, aggregate_element_list, aggregate_element_list,* and *aggregate exclude_list* are associated with the named object of the command.

The following arguments can determine the *effective element list*:

- a) -elements *element_list* adds the rooted names in *element_list* to the *aggregate_element_list*. It is not an error for an element to appear more than once in this list.
- b) **-model** *model_name* adds the rooted name of each instance that is an instance of the model to the *aggregate_element_list.*
- c) -models model_list or -model model_list adds the rooted name of each instance that is an instance of any of the models in model_list to the aggregate_element_list. It is not an error for a model to appear more than once in this list.
- d) -lib *lib_name* selects all models from the specified *lib_name*. If only -lib *lib_name* is specified, the rooted name of each instance that is an instance of every model present in *lib_name* is added to the *aggregate element list*.
- e) If -lib *lib_name* is specified along with -model *model_name* or -models *model_list*, the model is selected only if it is present in *lib_name*. This results in rooted names for only those models that are present in the *lib_name* library.
- f) If -lib *lib_name*, -model, or -models is specified with an -elements option, the *aggregate_element_list* is constructed by adding the rooted names from -elements and rooted names resulting from any -lib/-model/-models options.
- g) **-exclude_elements** *exclude_list* adds the rooted names in *exclude_list* to the *aggregate_exclude_list*. It is not an error for an element to appear more than once in this list. It is not an error for an element in the exclude list to not be in the *aggregate_element_list*.

- h) When -elements *element_list* is specified with a period (.), the current scope is included as a rooted instance in the *aggregate element list*.
- i) It shall be an error if the *element_list* is not specified as one of {}, {.}, or {*list*}.
- j) When **-transitive** is specified with the (default or explicit) value **TRUE**, elements (see <u>5.10.1</u>) in *aggregate_element_list* that are not leaf cells are processed to include the child elements (see <u>5.10.2</u>).
- k) The *prefilter_element_list* comprises the *aggregate_element_list* with any matching elements from the *aggregate_exclude_list* removed (see <u>5.10.2</u>).
- 1) The command arguments identified as filters are predicates that shall be satisfied by elements in the *effective_element_list*. The *prefilter_element_list* is filtered by the predicates to produce the *effective_element_list* (see 5.10.2).
- m) The range of legal element types is command dependent for each command that uses **-elements**. Each command specifies the effect of an empty *aggregate_element_list*. An explicitly empty list may be specified with {}.

5.10.1 Transitive TRUE

The detailed semantics of **-transitive TRUE** are described using <u>Figure 3</u>, <u>Figure 4</u>, and <u>Figure 5</u>. The figures are exemplary; the text provides a semantic for the validation of the result.

a) Given a design as shown in <u>Figure 3</u> with a instance A in the current scope, where A has child elements B, C, and D; B has child elements E and F, C has child elements G and H, and D has child elements I and J.





b) If the specification:

-elements {A A/C/H} -exclude_elements {A/C A/D} -transitive TRUE is applied to the design fragment shown in Figure 3, then Figure 4 shows the four specified elements by indicating them as boxed; those specified with exclude are shown with strike-through text.



Figure 4—Element processing specification

c) Figure 5 shows the results of the *effective_element_list*. The list includes $\{A \ A/B \ A/B/E \ A/B/F \ A/C/H\}$

The elements included or excluded by transitivity are shown as dashed-boxes or with strike-through text, respectively.



Figure 5—Element processing result

5.10.2 Result

The required result is derived as follows:

```
Begin // at the current scope.
Initialize by traversing the hierarchy and set element.mark := exclude
For each element in the aggregate_element_list do
    set element.mark := includeP
    if (transitive = TRUE AND element NOT Leaf_Cell) then
        foreach child in element call mark_child(child, include)
    end if
    done
For each element in the aggregate_exclude_list do
        set element.mark := excludeP
```

```
if (transitive = TRUE AND element NOT Leaf Cell) then
          foreach child in element call mark child(child, exclude)
      end if
   done
   For each element in the aggregate element list call check and add(element)
done
proc mark_child(element, value)
   if (element.mark != excludeP AND element.mark != includeP ) then
       element.mark := value
      if (element NOT Leaf Cell) then
          foreach child in element call mark child(child, value)
      end if
   end if
end proc
proc check and add(element)
   if (element.mark = includeP OR element.mark = include) then
      if (for all filters filter(element) = TRUE) then
          add element to effective element list
          if (transitive = TRUE AND element NOT Leaf Cell) then
              foreach child in element call check and add(child)
          end if
       end if
   end if
end proc
```

NOTE—Implementations may use any data structure or algorithm that produces the same results as the preceding method.

5.11 Command refinement

Some UPF commands support incremental refinement. Commands that support incremental refinement are called *refinable commands*. A refinable command may be invoked multiple times on the same object and each invocation may add additional arguments to those specified in previous invocations. The arguments of a refinable command that may be added after the first invocation are called *refining arguments*; these are shown in **boldface-green text** and labeled with an **R** in their respective *arguments* listings. Certain commands have refinable arguments; such arguments may have additional information about that argument added after the first invocation of the command, in much the same way that refinable commands may have additional arguments added later.

The first instance of a refinable command identifies the object to which it applies; all mandatory arguments shall be declared in this call and any other arguments may also be included. Subsequent occurrences of the command that identify the same object shall be executed in the same scope and shall include the **-update** option and refining arguments as required. The mandatory arguments that identify the object to which the command applies (the object name following the command or option name, and for strategies, the domain specification as well) shall also be included in each subsequent occurrence, but other mandatory arguments are not required in subsequent occurrences of the command. The end result will be as if all of the arguments, other than the **-update** argument, had been included in the initial occurrence of the command, either individually (e.g., **-clamp_value** or **-isolation_supply_set**) or merged together into a single argument (e.g., **-elements** or **-exclude_elements**).

For example, the **set_isolation** command (see <u>6.41</u>) can be invoked for the first time in a given scope to define a strategy name for a particular domain. Subsequent **set_isolation** commands executed in the same scope can specify the same strategy and domain names and also specify additional arguments to further characterize the isolation strategy defined by the previous command. Similarly, the **add_power_state**

command (see <u>6.4</u>) can be invoked initially in a given scope to define a set of power states for a supply set. A subsequent invocation of **add_power_state** in the same scope and for the same supply set may use the **-update** option to add a **-simstate** specification to each power state definition.

When -update is used for command refinement, the following apply:

- It shall be an error if -update is specified on the first command of a given kind that applies to a given object.
- It shall be an error if -update is not specified on subsequent commands of the same kind that apply to the same object.
- Except for those command arguments that aggregate (see <u>5.10</u> and <u>6.4</u>), it shall be an error if subsequent commands specify a value for a given argument that conflicts with or contradicts a previously specified value for the same argument.

Example

This shows a multiple-part refinement for a usage of set_isolation (see 6.41).

a) Constraint specification using port attributes

```
set_port_attributes
  -elements {a b c d}
  -clamp_value 0
```

b) Logical configuration

```
set_isolation demo_strategy -domain pda
  -elements {a b c d}
  -isolation_signal {iso_en}
  -isolation_sense {LOW}
```

c) Adding elements to the strategy

```
set_isolation demo_strategy -domain pda -update
   -elements {e f g}
```

d) Supply set implementation

```
set_isolation demo_strategy -domain pda -update
    -isolation_supply_set pda_isolation_supply
```

The implementation-independent part of the power intent [see item \underline{a})] could also be declared in the SystemVerilog HDL using the following attributes:

```
(* UPF_clamp_value = "0" *) out a;
(* UPF_clamp_value = "0" *) out b;
(* UPF_clamp_value = "0" *) out c;
(* UPF_clamp_value = "0" *) out d;
```

In this case, the declaration shall have identical semantics to the equivalent UPF command.

5.12 Error handling

If an error condition occurs, e.g., an incorrect command-line option is specified, then a TCL_ERROR exception shall be raised. This exception can be caught using the Tcl catch command, so these errors can be prevented from aborting the active **load_upf** command (see <u>6.28</u>). These errors shall have no impact on further commands. Processing may continue after the error is caught. Sequencing of the error catch and the choice of continuation is tool-dependent. The state of the design after an error is not defined. Specifically, a command that raises an error may partially complete before aborting.

In general, all commands that fail shall raise a TCL_ERROR. As described in the Tcl documentation, the global variables accessible after an error occurs include *errorCode* and *errorInfo*.

NOTE—The message string returned by the Tcl catch command is not specified in this standard.

5.12.1 errorCode

After an error has occurred, this variable contains additional information about the error in a form that is easy to process with programs. *errorCode* consists of a Tcl list with one or more elements. The first element of the list identifies a general class of errors and determines the format of the rest of the list. There are several formats for *errorCode* used by the Tcl code; see also the *Tcl command reference* [B6].

Errors defined in this standard are prefixed with UPF_, as shown in the following definitions. Individual applications that implement this standard may define and use additional error codes that do not start with UPF. Implementations need to use errors appropriate to their application.

a) UPF RETURN NOT VISIBLE error data

This error code indicates the objects referenced in the response of a query are not in the current scope. Queries return object names rooted in the current scope. Because they are called from a current scope that may be different from the scope in which all objects to be returned are visible, it shall be an error if the query cannot represent the objects to be returned as a rooted name.

The UPF_RETURN_NOT_VISIBLE error may be raised in these cases where there are no other errors. When this code is returned, the error_data is defined to be the same as the query would have returned, but with fully qualified names for the objects not visible in the current scope.

b) UPF_QUERY_OBJECT_NOT_DEFINED error_data

This error code indicates a query is called with a specific name argument and the named object is not defined in the current scope.

c) UPF_UPDATE_CONFLICT error_data

This error code indicates a command has been called with arguments that conflict with previously specified values.

d) UPF_UPDATE_MISSING error_data

This error code indicates a command has been called without the **-update** argument and the named object has already been defined.

e) UPF_UPDATE_OBJECT_NOT_FOUND error_data

This error code indicates a command has been called with the **-update** argument and the named object has not been previously defined.

f) UPF_OBJECT_NOT_FOUND error_data

This error code indicates a name referenced in a command is not defined in the current scope.

5.12.2 errorInfo

See the *Tcl command reference* [B6].

5.13 Units

Voltage values are expressed as real number literals that represent voltage measurements with the implicit unit of 1 V. For example, the literal 1.3 represents 1.3 V, or equivalently 1300 mV, or 1 300 000 μ V.

6. Power intent commands

This clause documents the syntax for each UPF command. For details concerning the simstate semantics, see <u>Clause 9</u>.

6.1 Categories

Each command in this clause is categorized based on the following definitions. Unless otherwise mentioned, all *constructs* (commands and/or options) in this standard are considered *Current*. Constructs considered as *Legacy* or *Deprecated* shall be explicitly denoted.

- a) *Current*—A construct defined in the standard with the following characteristics:
 - 1) It is recommended for use.
 - 2) Its semantics fully support the latest concepts.
 - 3) Its interaction with other related constructs is well defined.
 - 4) It is expected to be part of the standard and be considered for extension in future versions.
- b) Legacy—A construct defined in the standard with the following characteristics:
 - 1) It is *not recommended* for use for <u>new code</u>.
 - 2) Its semantics are not interoperable with all of the latest UPF concepts.
 - 3) It will not be considered for extensions in future versions.
 - It is included for backward compatibility only, e.g., set_isolation -isolation_power_net (see <u>6.41</u>).

Legacy constructs (commands and/or options) have not had their syntax and/or semantics updated to be consistent with other commands in this version of the standard, so their descriptions may contain significant obsolete information and their semantics may not be interoperable with the latest UPF concepts.

- c) Deprecated—A construct defined in the standard with the following characteristics:
 - 1) It is *not recommended* for use for <u>any code</u>.
 - 2) It will not be considered for extensions in future versions.
 - 3) It may be deleted from future versions, e.g., merge_power_domains (see <u>6.34</u>).

Deprecated commands are noted in this standard without syntax definitions or semantic explanations. Deprecated options of Current commands are noted in the syntax definition of those commands, but are not mentioned in the semantic explanations of those commands. For more details on any deprecated constructs, see IEEE Std 1801TM-2009 [B3].

For recommendations on how to use Current constructs to replace Legacy and Deprecated ones, see <u>Annex D</u>.

6.2 add_domain_elements [deprecated]

This is a deprecated command; see also 6.1 and Annex D.

6.3 add_port_state [legacy]

Purpose	Add states to a port	
Syntax	<pre>add_port_state port_name {-state {name <nom max="" min="" nom="" off="" ="">}}*</nom></pre>	
	port_name	The name of the supply port. Hierarchical names are allowed.
Arguments	-state {name <nom <br=""> min max min nom max off>}</nom>	The <i>name</i> and value for a state of the supply port. The value can be a nominal voltage; a pair specifying the minimum and maximum voltage; a triplet of values specifying the minimum, nominal, and maximum voltages; or off .
Return value	Return the fully qualified any of the port states are r	name (from the current scope) of the created port or raise a TCL_ERROR if not added.

This is a legacy command; see also 6.1 and Annex D.

The **add_port_state** command adds state information to a supply port. If the voltage values are specified, the supply net state is **FULL_ON** and the voltage value is the single nominal value or within the range of min to max; otherwise, if **off** is specified, the voltage value is **OFF**.

It shall be an error if *port_name* does not already exist.

It shall be an error if *nom* < *min* or *max* < *nom*.

Syntax example:

```
add_port_state VN1
   -state {active_state 0.88 0.90 0.92}
   -state {off state off}
```

6.4 add_power_state

Purpose	Define power state(s) of a power domain or supply set
Syntax	<pre>add_power_state object_name [-supply -domain] [-state {state_name [-supply_expr {boolean_expression}] [-logic_expr {boolean_expression}] [-simstate simstate] [-legal -illegal]}]* [-complete] [-update]</pre>

IEEE Std 1801-2013 IEEE STANDARD FOR DESIGN AND VERIFICATION OF LOW-POWER INTEGRATED CIRCUITS

	object_name	Simple name of a power domain or supply set.	
	-supply -domain	These arguments specify the kind of object to which this command applies. If -supply is specified, the <i>object_name</i> shall be the name of a supply set or a supply set handle. If -domain is specified, the <i>object_name</i> shall be the name of a power domain. If neither is specified, the type of <i>object_name</i> determines the kind of object to which the command applies.	
	-state {state_name}	<i>state_name</i> is the simple name of the state being defined or refined.	
Arguments	-supply_expr {boolean_expression}	-supply_expr specifies a Boolean expression defined in terms of supply ports, supply nets, and/or supply set handle functions that evaluates to <i>True</i> when the object is in the state being defined.	R
	-logic_expr {boolean_expression}	-logic_expr specifies a Boolean expression defined in terms of logic nets and/or power states of supply sets and/or power domains that evaluates to <i>True</i> when the object is in the state being defined.	R
	-simstate simstate	-simstate specifies a simstate for the power states associated with a supply set. Valid values are NORMAL, CORRUPT_ON_CHANGE, CORRUPT_STATE_ON_CHANGE, CORRUPT_STATE_ON_ACTIVITY, CORRUPT_ON_ACTIVITY, CORRUPT, and NOT_NORMAL. See <u>4.4.2.6</u> .	R
	-legal -illegal	These options specify the legality of the state being defined as either legal or illegal. The default is -legal.	R
	-complete	Specifies that all power states to be defined for this object have been defined. This implies that all legal power states have been defined and any state of the object that does not match a defined state is an illegal state.	R
	-update	Indicates this command provides additional information for a previous command with the same <i>object_name</i> and executed in the same scope.	R
Return value	Return an empty string i	f successful or raise a TCL_ERROR if not.	

Semantics

add_power_state defines one or more power states of an object. Each power state definition is independent of any other power state definition. Two different power states of the same object may have intersecting or overlapping -supply_expr and/or -logic_expr expressions. Such states may have different legalities. A power domain or a supply set may be in a state that matches more than one power state definition.

Multiple power states can be defined for an object in a single call to this command.

The power states defined for a given object include only those defined explicitly for that object [and for power states of a supply set or supply set handle, the default power states **DEFAULT_NORMAL** and **DEFAULT_CORRUPT** (see 4.6.3)]. Power states defined for one object are not inherited implicitly by any related object (e.g., by a supply set handle with which a supply set has been associated or vice versa). However, power states of one object may be defined in terms of power states of another object, to represent dependencies or correlation of power states.

The set of power states for a given object may be specified incrementally by using **-update**. The first **add_power_state** for that object may define one or more power states. Subsequent **add_power_state -update** commands for the same object may define additional power states.

A power state definition itself may also be specified incrementally by using **-update**. The initial definition of the power state defines at least the power state name and may specify additional information about this power state. Subsequent **add_power_state -update** commands for the same power state of the same object may specify additional details about that power state.

If a power state definition defined with a **-supply_expr** is updated with another **-supply_expr**, the definition becomes the conjunction of the two.

supply expr' = (*previous supply expr*) && (**-update** *supply expr*)

Similarly, if a power state definition defined with a -logic_expr is updated with another -logic_expr, the definition becomes the conjunction of the two.

logic_expr' = (previous logic_expr) && (-update *logic_expr)*

A logical contradiction exists when a logic net or supply set or power domain is specified to be more than one value for the state, e.g., (enable == 11') and (enable == 10'). A power state definition is *erroneous* if it contains logical contradiction(s).

The **-logic_expr** *boolean_expression* shall be a Boolean expression (see <u>5.4</u>) referencing control signals, clock signal intervals, and/or power states of a supply set or power domain. For convenience, the following expression forms may appear in this expression:

a) *interval(signal_name edge1 edge2)*

Equivalent to

the time between the most recent two specified edges of *signal_name* (returns the largest supported time value until both edges have occurred)

where

edge1, edge2 shall be one of posedge or negedge.

b) interval(signal_name edge)

Equivalent to *interval(signal_name edge edge)*

c) interval(signal_name)

Equivalent to *interval(signal_name* **posedge posedge**)

d) *supply_set* == *power_state*

Equivalent to { logic expression && supply expression }

where

logic_expression and *supply_expression* are the *boolean_expressions* used to define the *power_state* of *supply_set*.

e) *power_domain == power_state*

```
Equivalent to { logic expression }
```

where

logic_expression is the *boolean_expression* used to define the *power_state* of *power_domain*.

Examples

```
-logic_expr { enable == 1'b1 && interval(clk) < 5ps }
-logic_expr { core_pd.primary == ON_1d2v }
-logic_expr { core_pd == turbo && ram_pd != sleep }</pre>
```

Within a logic expression specified as part of a power state definition for a given power domain, the supply set handles of that power domain may be referenced directly without prefixing the name with the supply set or supply set handle name. To refer to an object declared in the current scope with the same name as a supply set handle of the power domain, the object name shall be prefixed with . /.

The **-supply_expr** *boolean_expression* shall be a Boolean expression (see <u>5.4</u>) that may reference supply nets, supply ports, and/or functions of supply sets or supply set handles. For convenience, the following expression forms may appear in this expression:

f) *supply net == net state*

```
Equivalent to
```

{ supply net.state == net state }

where

supply_net is the name of a supply port or net or a supply set (handle) function *net state* is the name of a state associated with *supply net*.

g) supply_net == { net_state min_voltage max_voltage }

```
Equivalent to
```

```
{ supply_net.state == net_state &&
  min_voltage <= supply_net.voltage && supply_net.voltage <= max_voltage }
where</pre>
```

supply_net is the name of a supply port or net or a supply set (handle) function (see <u>4.6.1</u>) *net_state* is the name of a state associated with *supply_net*.

h) supply_net == { net_state nom_voltage }

```
Equivalent to
```

{ supply_net == { net_state min_voltage max_voltage }} for verification.

where

```
supply_net is the name of a supply port or net or a supply set (handle) function
net_state is the name of a state associated with supply_net
min_voltage = nom_voltage - threshold
max_voltage = nom_voltage + threshold
threshold = 0.000001 * 10^(6 - min(6,sigdigits)) / 2
sigdigits = # of significant digits to the right of the decimal point in nom_voltage.
```

This form is for verification only; it is an error if it is used for implementation.

It is an error if *min_voltage > max_voltage*.

NOTE 1—The value of the Tcl variable tcl_precision, which specifies how many digits of precision are preserved when converting a floating-point number to a string, may affect the result of the preceding transformation if it is set to a number less than *sigdigits*.

i) supply_net == { net_state min_voltage nom_voltage max_voltage }

Equivalent to

{ *supply_net* == { *net_state min_voltage max_voltage* }} for verification. Implementation tools may use all three values to help make implementation choices.

It is an error if *min_voltage > nom_voltage* or *nom_voltage > max_voltage*.

Examples

{VDD == {	FULL_ON 0.8 } } is equivalent to { VDD == { FULL_ON 0.75 0.85 }}	
{Pwr == {	FULL_ON 0.925 } } is equivalent to { Pwr == { FULL_ON 0.9245 0.9255 }}	}
{Gnd == {	FULL ON 0.00 } } is equivalent to { Gnd == { FULL ON $-0.005 0.005 }$ }	ł

Within a supply expression specified as part of a power state definition for a given supply set or supply set handle, the functions of that supply set or supply set handle may be referenced directly without prefixing the name with the power domain name. To refer to an object declared in the current scope with the same name as a function of the supply set or supply set handle, the object name shall be prefixed with ./.

Restrictions

- j) If a supply expression is used to define a power state of a given supply set or supply set handle, it shall only refer to supply ports, supply nets, and/or functions of the given supply set or supply set handle. It is an error if such a supply expression refers to functions of another supply set or supply set handle.
- k) If a logic expression is used to define a power state of a given supply set or supply set handle, it shall only refer to logic ports, logic nets, interval functions, and/or power states of the given supply set or supply set handle. It is an error if such a logic expression refers to functions of a supply set or supply set handle, power states of another supply set or supply set handle, or power states of a domain.
- If a logic expression is used to define a power state of a given power domain, it shall only refer to logic ports, logic nets, interval functions, power states of supply sets or supply set handles, and/or power states of other power domains. It is an error if such a logic expression refers to supply ports, supply nets, or functions of a supply set or supply set handle.
- m) It is an error if a supply expression is used to define a power state of a power domain.
- n) It is an error if a simstate is associated with a power state of a power domain.
- o) When -simstate
 - 1) is first specified for a named state, any of the arguments may appear.
 - 2) is specified as **NOT_NORMAL**, the effect shall be the same as if **CORRUPT** had been specified, see (4.6.3), except that the definition may be subsequently refined to any simstate other than **NORMAL**.
 - 3) has previously been specified as NORMAL, CORRUPT, CORRUPT_ON_ACTIVITY, CORRUPT_ON_CHANGE, CORRUPT_STATE_ON_CHANGE, or CORRUPT_STATE _ON_ACTIVITY, it shall be an error if an add_power_state -update command for the same object specifies any simstate other than that originally specified (e.g., once CORRUPT has been specified for a particular state, it shall remain as CORRUPT in any subsequent updates for the definition of that state).
- p) The simstate for **DEFAULT_NORMAL** is **NORMAL**.
- q) The simstate for **DEFAULT_CORRUPT** is **CORRUPT**.
- r) There is no default simstate for a user-defined power state.
- s) The supply set is in the **DEFAULT_CORRUPT** power state when it is not in one of the defined power states of the supply set that have simstates defined on them, including the **DEFAULT_NORMAL** predefined state.
- t) If **-illegal** has been specified in the definition of a power state for a given object, it is an error if that object is in a state that matches the definition of that power state. A verification tool shall emit an error message when an object is in an illegal power state.
- u) If **-complete** has been specified in an **add_power_state** command for a given object, it is an error if that object is in a state that does not match any of the defined power states. A verification tool shall emit an error message when an object is in such an undefined state.
v) If **-complete** has been specified on an **add_power_state** command for a given object, it is an error if a subsequent update to that command is executed.

NOTE 2-The choice of state name has no simstate implications.

NOTE 3—Implementation tools may optimize a design based on the presumption illegal states never occur. Such optimizations are allowed only if they do not change the behavior of the design.

NOTE 4—If the **add_power_state** command for the primary supplies of two interconnected domains are both defined as complete, this implies that all intended legal states have been defined for each domain, and, therefore, all possible state combinations of the two domains have been defined.

Syntax examples:

```
add power state PdA.primary -supply
-state {GO MODE
   -logic expr SW ON -simstate NORMAL
   -supply expr { (power == {FULL ON 0.8})
   && (ground == \{FULL ON 0\})
   && (nwell == {FULL ON 0.8}) }
-state {OFF MODE
   -logic expr {!SW ON}
   -supply expr {power == {OFF}}
   -simstate CORRUPT}
-state {SLEEP MODE
   -logic expr {SW ON && (interval(clk dyn posedge negedge) >= 100ns)}
   -supply_expr { (power == {FULL ON 0.8})
   && (ground == {FULL ON 0})
   && (nwell == {FULL ON 1.0}) }
   -simstate CORRUPT STATE ON CHANGE }
add power state PdA.primary -supply -update -complete
add power state PdTOP -domain
   -state {GOGO -logic_expr {u1/PdA.primary == GO MODE}}
add power state PdTOP -state {GOGO -legal} -update
```

6.5 add_pst_state [legacy]

Purpose	Define the states of each of	f the supply nets for one possible state of the design
Syntax	add_pst_state state_name -pst table_name -state supply_states	2
	state_name	The simple name of the state being defined.
Arguments	-pst pst_name	The power state table (PST) to which this state applies.
Aiguments	-state supply_states	The list of supply net state names (see <u>6.20</u>), listed in the corresponding order of the -supplies listing in the create_pst command (see <u>6.19</u>). A * in place of a state name indicates this is a "don't care" for that supply.
Return value	Return a 1 if successful or	raise a TCL_ERROR if not.

This is a legacy command; see also 6.1 and Annex D.

The **add_pst_state** command defines the name for a specific state of the supply nets defined for the PST *table name*.

It shall be an error if

- The number of *supply states* is different from the number of supply nets within the PST.
- A state name is defined more than once for the same PST.

Syntax example:

create_pst	pt -supplies	{	PN1	PN2	SOC/OTC/PN3	}
add_pst_state s1 -pst	pt -state	{	s08	s08	s08	}
add_pst_state s2 -pst	pt -state	{	s08	s08	off	}
add pst state s3 -pst	pt -state	{	s08	s09	off	}

6.6 apply_power_model

Purpose	Connects the interface supply set handles of a previously loaded power model		
Syntax	<pre>apply_power_model power_model_name [-elements instance_list] [-supply_map {{lower_scope_handle upper_scope_supply_set}*}]</pre>		
	power_model_name	The name a previously defined power model. See 6.8 .	
	-elements instance_list	The list of instances to which the specified power model applies.	
Arguments	<pre>-supply_map {{lower_scope_handle upper_scope_supply_set }*}</pre>	How the interface supply handles of the corresponding power model con- nect with the actual supply sets or supply set handles in the current scope.	
Return value	Return a 1 if successful or	r raise a TCL_ERROR if not.	

The **apply_power_model** command describes the connections of the interface supply set handles of a previously loaded power model with the supply sets in the scope where the corresponding macro cells are instantiated.

If **-elements** is not specified, the specified supply association is applied to all instantiations of targeted macro cells by the specified power model (see 6.8) under the current scope. The general precedence rules in 5.8 apply here as well.

Each pair in the **-supply_map** option implies an **associate_supply_set** command (see <u>6.7</u>) of the following general form:

```
associate_supply_set upper_scope_supply_set
-handle lower scope handle
```

The arguments of the **-supply_map** option need to be such that the implied **associate_supply_set** commands are legal.

The supply connection specified by **-supply_map** overwrites any implicit or automatic supply set connection. It is an error if a specified supply connection in **-supply_map** conflicts with any explicit connections.

The following also apply:

a) It is an error to update any power intent command specified within a power model from outside of this model.

If a power model has any power state specified with simstate **NOT_NORMAL**, it cannot be updated with a specific simstate from commands outside of the power model. As a result, the **CORRUPT** simulation semantics shall apply to the power state (see <u>9.4</u>).

- b) The processing of this command shall follow the description in <u>Clause 8</u>.
- c) When apply_power_model is used with -elements, it is an error if the underlying cell name of each instance does not match the corresponding macro cell name specified in the -for option of begin_power_model (see <u>6.8</u>) or the power_model_name when the -for option (of begin_power_model) is not specified.

Syntax example:

```
apply_power_model upf_model -elements I1 -supply_map {{PD.ssh1 ss1} {PD.ssh2 ss2}}
```

For other examples of using these commands, see <u>Annex E</u>.

6.7 associate_supply_set

Purpose	Associate a supply set wi	ith a power domain, power switch, or strategy supply set handle	
Syntax	associate_supply_set supply_set_name -handle supply_set_handle		
	supply_set_name	The rooted name of a supply set to associate with a supply set handle.	
Arguments	-handle supply_set_handle	The supply set handle with which the supply set is associated.	
Return value	Return an empty string if	f successful or raise a TCL_ERROR if not.	

The **associate_supply_set** command associates a supply set with a power domain, power switch, or strategy supply set handle (see <u>5.3.3.2</u>). As a result, each function of the named supply set is associated with the corresponding function of the supply set handle, which makes the named supply set and the supply set handle equivalent (see <u>4.4.3</u>). Both the *supply_set_name* and the *supply_set_handle* shall refer to predefined or previously created supply sets.

The *supply_set_handle* may be a predefined supply set handle. The predefined supply set handles are as follows:

- a) The predefined supply set handles for a power domain *domain_name* (see <u>6.17</u>) include: *domain_name.primary*, *domain_name.default_retention*, and *domain_name.default_isolation*. User-defined names for *supply set handle* are also permitted.
- b) The predefined supply set handle for a power-switch *switch_name* (see <u>6.18</u>) is *switch_name*.supply.
- c) The predefined supply set handles for an isolation cell strategy *isolation_name* (see <u>6.41</u>) are *domain_name.isolation_name.isolation_supply_set*, if there is only one isolation supply set, or *domain_name.isolation_name.isolation_supply_set[index]*, where *index starts* at 0, if there are multiple isolation supply sets. The named supply set may be associated with one of these supply set handles using the **associate_supply_set** command as follows:

```
associate_supply_set U1/PD1.my_iso.isolation_supply_set\[1\]
    -handle U1/PD1.my iso.clamp
```

- d) The predefined supply set handles for a level-shifter strategy *level_shifter_name* (see <u>6.43</u>) are *domain name.level shifter name.input* and *domain name.level shifter name.output*.
- e) The predefined supply set handle for a retention strategy *retention_name* (see <u>6.49</u>) is *domain name.retention name.supply*.

It shall be an error if

- The supply set handle is defined for a strategy and more than one supply set is associated with that supply set handle.
- The supply set handle is defined for a power domain, and more than one supply set defined in an ancestor scope is associated with that supply set handle, or more than one supply set defined in the scope of the power domain or a descendant scope is associated with that supply set handle.
- The associations of supply sets with supply set handles form a loop of associations.

Syntax examples:

```
associate_supply_set some_supply_set
-handle U1/PD1.mem ss
```

NOTE—A supply set handle can also appear as the *supply_set_name* in an **associate_supply_set** command. This allows transitive association of supply sets, such as the following:

```
associate_supply_set top_level_SS -handle PD1.primary
associate_supply_set PD1.primary -handle PD2.backup
associate supply set PD1.primary -handle PD3.default isolation
```

6.8 begin_power_model

Purpose	Define a power model	
Syntax	begin_power_model power_for model_list]	ver_model_name
Arguments	power_model_name	The name of the power model.
	-for model_list	The names of the hard IP or macro cells to which the power model applies.
Return value	Return a 1 if successful o	r raise a TCL_ERROR if not.

The **begin_power_model** and **end_power_model** (see <u>6.25</u>) commands define a power model containing other UPF commands. A power model is used to define the power intent of a hard IP and shall be used in conjunction with one or more model representations. A power model defined with **begin_power_model** is terminated by the first subsequent occurrence of **end power model** in the same UPF file.

-for indicates that the power model represents the power intent for a family of model definitions. When -for is not specified, the *power_model_name* shall also be a valid macro cell name. It is an error if the targeted model has a UPF_is_leaf_cell attribute set to FALSE. It is also an error if any design objects referred to in a power model cannot be found in the corresponding library model or behavioral model of the cell.

A power model can be referenced by its simple name from anywhere in a power intent description. It is an error to have two power models with the same name.

It is also an error if the following commands are specified within the model:

- name_format (see <u>6.35</u>)
- save_upf (see $\underline{6.36}$)
- set_scope (see <u>6.52</u>)
- load_upf -scope (see <u>6.28</u>)
- **begin_power_model** (see <u>6.8</u>)
- apply_power_model (see <u>6.6</u>)
- Any deprecated/legacy commands/options (see <u>Annex D</u>)

To specify supplies coming into or out of the model, or a supply that has at least one data port related to it, use the **-supply** option of the **create_power_domain** command (see <u>6.17</u>) for the top-scope power domain of the power model. In addition, the system power states defined upon these supply set handles become the power state definition at the interface of the power model, which shall be consistent with the upper-scope system power states into which the corresponding upper-scope supply sets are mapped (see <u>6.6</u>). The defined supply set handles are also called *interface supply handles* of the power model. Finally, the simstate simulation semantics described in <u>9.4</u> applies to all supply sets or supply set handles defined within a power model.

All power commands within a power model describe power intent that has already been implemented with the targeted cells. No new logic or design objects shall be inferred within the cell instances targeted by a power model.

A power model can be applied to specific instances using **apply_power_model** (see <u>6.6</u>). A power model that is not referenced by an **apply_power_model** command does not have any impact on the power intent of the design.

Syntax example:

```
begin_power_model upf_model -for cellA
    create_power_domain PD1 -elements {.} -supply {ssh1} -supply {ssh2}
    ;# other commands ...
end power model
```

For more examples of using these commands, see <u>Annex E</u>.

6.9 bind_checker

Purpose	Insert checker modules and bind them to instances
Syntax	<pre>bind_checker instance_name -module checker_name [-elements element_list] [-bind_to module [-arch name]] [-ports {{port_name net_name}*}]</pre>

	instance_name	The name used to instance the checker module in each instance.
	-module checker_name	The name of a SystemVerilog module containing the verification code. The verification code itself shall be written in SystemVerilog, but it can be bound to either a SystemVerilog or VHDL instance.
Arguments	<pre>-elements element_list</pre>	The list of instances.
	<pre>-bind_to module [-arch name]</pre>	The SystemVerilog module or VHDL entity/architecture for which all instances are the target of this command.
	<pre>-ports {{port_name net_name}*}</pre>	The association of signals to the checker's ports.
Return value	Return an empty string if	successful or raise a TCL_ERROR if not.

The **bind_checker** command inserts checker modules into a design without modifying the design code or introducing functional changes. The mechanism for binding the checkers to instances relies on the SystemVerilog bind directive. The bind directive causes one module to be instantiated within another, without having to explicitly alter the code of either. This facilitates the complete separation between the design implementation and any associated verification code.

Signals in the target instance are bound by position to inputs in the bound checker module through the port list. Thus, the bound module has access to any and all signals in the scope of the target instance, by simply adding them to the port list, which facilitates sampling of arbitrary design signals.

If **-bind_to** is specified, an instance of checker is created in every instance of the module. Otherwise, an instance of the checker is only created within the current scope.

port_name is a port defined on the interface of *checker_name* and *net_name* is a name of a net relative to the scope where *checker_name* is being instantiated.

It shall be an error if *instance name* already exists in **-bind_to** *module*.

This command is for verification only; implementation tools shall ignore it.

Syntax example:

```
bind_checker chk_p_clks
-module assert_partial_clk
-bind_to aars
-ports {{prt1 clknet2} {port3 net4}}
```

Modeling mutex assertions

To model mutex assertions (see 6.50 and 6.49), the assertions can be put in a SystemVerilog checker_module with following interface:

```
module checker_module ( save, restore, reset_a, clock_a );
input save, restore, reset_a, clock_a;
... different mutex assertions ...
endmodule
```

The bind_checker command would look like the following:

```
bind_checker mutex_checker_inst -module checker_module \
-ports { {save PDA.test_retention.save_signal } \
{ restore PDA.test_retention.restore_signal } \
{ reset_a reset_a } \
{ clock_a clock_a } }
```

6.10 connect_logic_net

Purpose	Connect a logic net to logic ports		
Syntax	<pre>connect_logic_net net_name -ports port_list [-reconnect]</pre>		
Arguments	net_name	A simple name.	
	-ports port_list	A list of ports on the interface of the current scope and/or on instances that are located in the current scope and its descendants.	
	-reconnect	Allows a port that is already connected to a net to be disconnected from the existing net and connected to <i>net_name</i> .	
Return value	Return an empty string	if successful or raise a TCL_ERROR if not.	

The **connect_logic_net** command connects a logic net to the specified ports. The net is propagated through implicitly created ports and nets throughout the logic hierarchy in the descendant subtree of the active UPF scope as required to support connections created by **connect_logic_net** (see <u>9.2</u>). The connection from *net_name* in the active UPF scope to any element in *port_list* shall not cross any power-domain boundaries.

The net and ports shall be of a compatible type. The following HDL types are compatible with each other:

- SystemVerilog logic
- VHDL std_ulogic

It shall be an error if:

- a) *net_name* is not the name of a logic net defined in the current HDL scope either explicitly or implicitly as a result of a **create_logic_net** command.
- b) A HighConn port in *port_list* is already connected to a different net than *net_name*, unless the **-reconnect** option is specified.
- c) A LowConn port in *port_list* is already connected to a different net than *net_name*.
- d) The same port name occurs in the *port_list* of multiple **connect_logic_net** commands with different *net name* arguments.

NOTE 1—Use create_logic_port (see <u>6.16</u>) to create new logic ports on power-domain boundaries.

NOTE 2—This command exists to allow for the propagation of signals from a power-management block. Using this command to provide non-power control connections may cause the logic function to diverge from the HDL and is strongly discouraged.

Syntax example:

```
connect_logic_net tree_top
  -ports {s b}
```

6.11 connect_supply_net

Purpose	Connect a supply net to supply ports			
Syntax	<pre>connect_supply_net net_name [-ports ports_list] [-pg_type {pg_type_list element_list}]* [-vct vct_name] [-cells cells_list] [-domain domain_name] [-pins pins_list] [-rail_connection_rail_type]</pre>			
	net_name	A simple name.		
	-ports ports_list	A list of rooted port names.		
	<pre>-pg_type {pg_type_list element_list}</pre>	An indirect connection specification via the <i>pg_type</i> on the instance's ports.		
Arguments	-vct vct_name	A value conversion table (VCT) defining how values are mapped from UPF to an HDL model or from the HDL model to UPF.		
	-cells cells_list	A list of cells to use for -pg_type or -rail_connection .		
	-domain domain_name	The domain indicates the scope to use for -pg_type or -rail_connection .		
Deprecated	-pins pins_list	A list of pins on cells to connect. This is a deprecated option; see also <u>6.1</u> and <u>Annex D</u> .		
arguments	-rail_connection	The rail type (for older libraries).		
	rail_type	This is a deprecated option; see also <u>6.1</u> and <u>Annex D</u> .		
Return value	Return an empty string if	successful or raise a TCL_ERROR if not.		

The **connect_supply_net** command connects a supply net to the specified ports. The net is propagated through implicitly created ports and nets throughout the logic hierarchy in the descendant subtree of the current scope as required to support supply port/net connections made explicitly, automatically, or implicitly (see 9.2) This explicit connection overrides (has higher precedence than) the implicit and automatic connection semantics (see 9.2) that might otherwise apply. **-domain** or **-cells** is required when the **-pg_type** option is specified.

If **connect_supply_net** is used to connect a supply net defined with create_supply_net -domain D (see <u>6.20</u>) to a pg pin of an instance, then the instance shall be in the extent of power domain D.

Use the following:

-ports to connect to supply ports.

-cells to connect to all pins of the appropriate type (power or ground) on the specified cells.

-pg_type to connect to ports on the instances that have the specified pg_type.

-vct to indicate that for every HDL port to which the net is connected, the supply net state shall be converted if it is being propagated into the HDL port (see 6.23) or the HDL port value shall be converted if it is being propagated onto the supply net (6.14). -vct is ignored for any connections of the supply net to supply ports defined in UPF.

The following also apply:

- It shall be an error if any cell, domain, port, supply net, or instance specified in this command does not exist.
- It shall be an error if the value conversions specified in the value conversion table (VCT) do not match the type of the HDL port.
- It shall be an error if neither **-ports** nor **-pg_type** is specified in a **connect_supply_net** command.
- The -ports option is mutually exclusive with the -cells, -domain, and -pg_type options.
- Automatic propagation of a supply net throughout the extent of a power domain is determined by its usage within the domain, such as primary supply, retention supply, etc.
- It shall be an error if *net_name* has not been previously created; in this case, a 0 shall be returned.
- If -pg_type is specified, it shall be an error if an instance does not exist or the specified attribute does not exist on any port of the instance.

Syntax examples:

```
connect_supply_net fb
-ports {jk jb}
connect_supply_net mc
-ports {rl}
-vct SV_TIED_HI
```

The following command connects the supply net VDDX to the VDD port of a hierarchical instance I1/I2:

connect_supply_net VDDX -ports I1/I2/VDD

The following command connects the supply net VDDX to the VDD ports of all instances within hierarchical instance I1/I2:

6.12 connect_supply_set

Purpose	Connect a supply set to pa	articular elements	
Syntax	<pre>connect_supply_set_supply_set_ref {-connect {supply_function pg_type_list}}* [-elements element_list] [-exclude_elements exclude_list] [-transitive [<true false="" ="">]]</true></pre>		
Arguments	supply_set_ref	The rooted name of the supply set.	
	-connect {supply_function pg_type_list}	Define automatic connectivity for a <i>supply_function</i> of the <i>supply_set_ref</i> as ports having the specified <i>pg_type_list</i> attributes (see <u>6.11</u>).	
	-elements element_list	The list of instance names to add.	
	-exclude_elements exclude_list	The list of instances to exclude from the <i>effective_element_list</i> .	

Arguments	-transitive [<true <br="">FALSE>]</true>	If -transitive is not specified at all, the default is -transitive TRUE . If -transitive is specified without a value, the default value is TRUE .
Return value	Return an empty string in	Successful or raise a TCL_ERROR if not.

The **connect_supply_set** command connects a supply set to the specified elements. The nets of the set are propagated through implicitly created ports and nets throughout the logic hierarchy in the descendant subtree of the current scope as required to implement the supply net connection (see <u>9.2</u>) This explicit connection overrides (has higher precedence than) the implicit and automatic connection semantics (see <u>9.2</u>) that might otherwise apply.

This command applies to elements in the *effective_element_list* (see <u>5.10</u>) as follows:

- a) When *supply_set_ref* refers to a handle associated with a domain, the *prefilter_element_list* is filtered to only include elements within the extent of the domain.
- b) When *supply_set_ref* refers to a handle associated with a strategy, the *prefilter_element_list* is filtered to only include all elements connected to the strategy's supply.
- c) When *supply_set_ref* refers to a handle associated with a domain and the *aggregate_element_list* is empty, all elements in the extent of the domain are added to the *aggregate element list*.
- d) When *supply_set_ref* refers to a handle associated with a strategy and the *aggregate_element_list* is empty, all elements connected to the respective strategy supply are added to the *aggregate_element_list*.

-connect is additive, i.e., on a particular supply function, a subsequent invocation setting *pg_type_list* adds the additional *pg_type_list*.

NOTE—The *exclude_list* in **-exclude_elements** can specify elements that have not already been explicitly or implicitly specified via an explicit or implied *element_list*.

It shall be an error if

- A particular pg_type_list is associated with more than one supply net for any given instance in -connect.
- More than one supply net is connected to the same port in an instance, even if the connection is the result of more than one command that connects supply nets, e.g, connect_supply_set, connect_supply_net, etc.
- Any element of *element_list* or *exclude_list* is not in a specified domain or strategy referenced in the supply set handle.

Syntax example:

```
connect_supply_set some_supply_set
  -elements {U1/U_mem}
  -connect {power {primary_power}}
  -connect {ground {primary ground}}
```

6.13 create_composite_domain

Purpose	Define a composite domain comprised of one or more subdomains			
Syntax	<pre>create_composite_domain composite_domain_name [-subdomains subdomain_list] [-supply {supply_set_handle [supply_set_ref]}] [-update]</pre>			
Arguments	composite_domain_ name	The name of the composite domain; this shall be a simple name.		
	-subdomains subdomain_list	The -subdomains option specifies a list of rooted domain names, including any previously created composite domains.	R	
	-supply {supply_set_handle [supply_set_ref]}	The -supply option specifies the <i>supply_set_handle</i> for <i>composite_domain_name</i> . If <i>supply_set_ref</i> is also specified, the domain <i>supply_set_handle</i> is associated with the specified <i>supply_set_ref</i> . The <i>supply_set_ref</i> may be any supply set visible in the current scope. See also <u>6.7</u> .	R	
	-update	Use -update if the <i>composite_domain_name</i> has already been defined.	R	
Return value	Return an empty string if	successful or raise a TCL_ERROR if not.		

A *composite power domain* is a simple container for a set of power domains. Unlike a power domain, a composite domain has no corresponding physical region on the silicon. Attributes like power states and the primary *supply_set_handle* can be specified on a composite domain, but these attributes shall not be applied to subdomains. However, operations performed on the composite domain shall be applied to each subdomain, e.g., defining a strategy.

The following commands, applied to a composite domain, are applied to each subdomain if and only if the application of that command does not result in an error in any subdomain:

connect_supply_net map_power_switch map_retention_cell set_isolation set_level_shifter set_repeater set_retention use interface cell

Only the primary supply handle can be specified in the **-supply** option. The following also apply:

- a) Composite power domains can be used as a subdomain of other composite power domains.
- b) Since a composite domain is simply a container, commands can still be applied to subdomains after composition.
- c) For each subdomain: If a supply set is associated with the primary *supply_set_handle* of a subdomain, that supply set shall be equivalent to the primary supply set of the composite domain or declared as equivalent to the primary supply set of the composite domain (see also <u>6.40</u>).

- d) Commands applied to a subdomain do not affect any other subdomain or the composite domain.
- e) Subdomains of a composite domain can still be referenced after composition, in the sense their elements lists are valid after composition, and all aspects of the subdomain (e.g., strategies defined on them) can be referenced.

When the primary *supply_set_handle* and a *supply_set_ref* are specified in **-supply**, it is equivalent to the following:

associate_supply_set supply_set_ref
 -handle composite_domain_name.primary

Syntax example:

```
create_composite_domain my_combo_domain_name
-subdomains {a/pd1 b/pd2}
-supply {primary could_be_on_ss}
```

6.14 create_hdl2upf_vct

Purpose	Define a VCT that can be used in converting HDL logic values into state type values	
Syntax	<pre>create_hdl2upf_vct vct_name -hdl_type {<vhdl sv="" =""> [typename]} -table {{from_value to_value}*}</vhdl></pre>	
	vct_name	The VCT name.
Arguments	-hdl_type { <vhdl sv="" =""> [typename]}</vhdl>	The HDL type for which the value conversions are defined.
	<pre>-table {{from_value to_value}*}</pre>	A list of the values of the HDL type to map to UPF state type values.
Return value	Return an empty string if	successful or raise a TCL_ERROR if not.

The **create_hdl2upf_vct** command defines a value conversion table (VCT) from an HDL logic type to the state type of the supply net value (see <u>Annex B</u>) when that value is propagated from HDL port to a UPF supply net. It shall provide a conversion for each possible logic value that the HDL port can have. **create_upf2hdl_vct** does not check that the set of HDL values are complete or compatible with any HDL port type.

 vct_name provides a name for the value conversion table for later use with the **connect_supply_net** command (see <u>6.11</u>). A VCT can be referenced by its simple name from anywhere in a power intent description. It is an error to have two VCTs with the same name. The predefined VCTs are shown in <u>Annex F</u>.

-hdl_type specifies the HDL type for which the value conversions are defined. This information allows a tool to provide completeness and compatibility checks. If the *typename* is not one of the language's predefined types or one of the types specified in the next paragraph, then it shall be of the form *library.pkg.type*.

The following HDL types shall be the minimum set of types supported. An implementation tool may support additional HDL types.

- a) VHDL
 - 1) Bit, std [u]logic, Boolean
 - 2) Subtypes of std [u]logic
- b) SystemVerilog

reg/wire, Bit, Logic

-table defines the 1:1 conversion from HDL logic value to the UPF partially on and on/off states. The values are consistent with the HDL type values.

For example:

- When converting from SystemVerilog *logic type*, the legal values are 0, 1, X, and Z.
- When converting from SystemVerilog or VHDL bit, the legal values are 0 or 1.
- When converting from VHDL std [u]logic, the legal values are U, X, O, 1, Z, W, L, H, and -.

The conversion values have no semantic meaning in UPF. The meaning of the conversion value is relevant to the HDL model to which the supply net is connected.

Syntax examples:

```
create hdl2upf vct
   vlog2upf vss
   -hdl type {sv reg}
   -table {{X OFF} {0 FULL ON} {1 OFF} {Z PARTIAL ON}}
create hdl2upf vct
   stdlogic2upf vss
   -hdl type {vhdl std logic}
   -table {{ 'U' OFF}
        { 'X' OFF }
        { '0' OFF }
        { '1' FULL_ON }
        { 'Z' PARTIAL ON }
        { `W' OFF }
        { `L' OFF }
        { 'H' FULL ON }
        { '-' OFF } }
```

6.15 create_logic_net

Purpose	Define a logic net
Syntax	<pre>create_logic_net net_name</pre>
Arguments	<i>net_name</i> A simple name.
Return value	Return an empty string if successful or raise a TCL_ERROR if not.

The **create_logic_net** command creates a logic net in the current scope or identifies a logic net in the current scope.

The net's type is determined by the language of the scope where it is created. If the scope is

- SystemVerilog, the type is logic
- VHDL, the type is std ulogic

NOTE—This command exists to allow for the propagation of signals from a power-management block. Using this command to provide non-power control connections may cause the logic function to diverge from the HDL and is strongly discouraged.

Syntax example:

```
create logic net iso ctrl
```

6.16 create_logic_port

Purpose	Define a logic port	
Syntax	create_logic_port port_name [-direction <in inout="" out="" ="">]</in>	
	port_name	A simple name.
Arguments	-direction <in out="" ="" <br="">inout></in>	The direction of the port. The default is in.
Return value	Return an empty string if successful or raise a TCL_ERROR if not.	

The **create_logic_port** command creates a logic port in the current scope. Logic ports are effectively created before isolation and level-shifting strategies are applied (see 4.3.3); therefore, any isolation or level-shifting strategy defined for a power domain may apply to logic ports created on the boundary of that power domain, regardless of the order in which the **create_logic_port** command and the **set_isolation** (see 6.41) or **set_level_shifter** (see 6.43) commands occur, provided the logic port matches the criteria specified in the strategy.

The port's type is determined by the language of the scope where it is created. If the scope is

- SystemVerilog, the type is logic
- VHDL, the type is std_ulogic

The created port is equivalent to a module port created in SystemVerilog or VHDL with the same name and direction. Logic ports are sources, sinks, or both.

- a) The LowConn of an input port is a source.
- b) The HighConn of an input port is a sink.
- c) The LowConn of an output port is a sink.
- d) The HighConn of an output port is a source.
- e) The LowConn of an inout port is both a source and a sink.
- f) The HighConn of an inout port is both a source and a sink.

NOTE—This command exists to allow for the propagation of signals from a power-management block. Using this command to provide non-power control connections may cause the logic function to diverge from the HDL and is strongly discouraged.

Syntax example:

```
create_logic_port test_lp
-direction out
```

6.17 create_power_domain

Purpose	Define a power domain and its characteristics		
Syntax	<pre>create_power_domain domain_name [-simulation_only] [-atomic] [-elements element_list] [-exclude_elements exclude_list] [-supply {supply_set_handle [supply_set_ref]}]* [-available_supplies supply_set_ref_list] [-define_func_type {supply_function pg_type_list}]* [-update] [-include_scope] [-scope instance_name]</pre>		
	domain_name	The name of the power domain; this shall be a simple name rooted in the current scope.	ne
	-simulation_only	Define a power domain for simulation purposes only.	
	-atomic	Define the minimum extent of the power domain.	
	-elements element_list	The list of instances to add.	R
Arguments	-exclude_elements exclude_list	The list of instances to exclude from the <i>effective_element_list</i> .	R
	-supply {supply_set_handle [supply_set_ref]}	The -supply option specifies the <i>supply_set_handle</i> for <i>domain_name</i> . If <i>supply_set_ref</i> is also specified, the domain <i>supply_set_handle</i> is associated with the specified <i>supply_set_ref</i> . The <i>supply_set_ref</i> may be any supply set visible in the current scope. The predefined <i>supply_set_handles</i> are: primary , default_retention , and default_isolation . See also <u>6.7</u> .	R
	-available_supplies supply_set_ref_list	A list of additional supply sets that are available for use by implemen- tation tools to power cells inserted in this domain.	
Arguments	-define_func_type {supply_function pg_type_list}	Define automatic connectivity for a <i>supply_function</i> of <i>domain_name</i> .primary (see <u>6.7</u>) having the specified attributes in <i>pg_type_list</i> .	R
	-update	Use -update if the <i>domain_name</i> has already been defined.	R
Deprecated arguments	-include_scope	Define the extent of the domain to include the current scope and, by def all of its descendant scopes. See also 5.8 . This is a deprecated option; see also 6.1 and <u>Annex D</u> .	àult,
	-scope instance_name	Create the power domain within this scope. This is a deprecated option; see also 6.1 and <u>Annex D</u> .	
Return value	Return an empty string i	f successful or raise a TCL_ERROR if not.	

create_power_domain defines a power domain and the set of instances that are in the extent of the power domain. It may also specify whether the power domain can be partitioned further by subsequent commands.

-elements specifies a set of rooted instances contained within the power domain. Although the syntax of this command does not include a -transitive option, its semantics are as if any occurrence of the command has the value -transitive TRUE (see <u>5.10.1</u>). The following also apply:

- *element_list* shall contain instance names rooted in the current scope.
- Each design top instance (see <u>4.2.7</u>) and each of its descendant instances shall be in the extent of exactly one power domain.
- When -simulation_only is specified, signal names and process labels may also be specified in *list*.
 -simulation_only specifies the domain is intended for use with behavioral non-synthesizing elements.
- When **-atomic** is specified, all elements originally included in the extent of the power domain shall always remain in the extent of that power domain.
- The power domain shall be created in the current scope.
- The -elements option shall be used at least once in the specification of a power domain using create_power_domain; this can be in the first invocation (i.e., without the -update option) or during the subsequent updates (i.e., with the -update option).
- If the value of *effective_element_list* (see <u>5.10</u>) is an empty list, a domain with the name *domain name* is created, but with an empty extent.
- If the value of the *effective_element_list* (see <u>5.10</u>) is a period (.), the current scope is included in the extent of the domain.

NOTE 1—A design top instance can be included in the extent of a power domain created in the scope of that instance by specifying -elements {.} in the **create_power_domain** command.

NOTE 2—If the current scope is set to instance i0, then create_power_domain PD -elements {.} would include the current scope (i0) and all of its descendants in the power domain PD. In contrast, create_power_domain PD -elements {i1 i2 ... ik} would not include i0 in the power domain, but would only include its descendants i1, i2, ..., ik. In either case, the scope of the power domain PD is the same, because in both cases the current scope was i0 when the **create_power_domain** command was executed.

An instance that has no parent or whose parent is in the extent of a different power domain is called a *boundary instance*.

The upper boundary of a power domain consists of

— the LowConn side of each port of each boundary instance in the extent of this domain.

The lower boundary of a power domain consists of

- the HighConn side of each port of each boundary instance in the extent of another power domain, where the parent of the boundary instance is in the extent of this domain, together with
- the HighConn side of each port of any macro cell instance in this power domain, for which the related supply set is neither identical to nor equivalent to the primary supply set of this domain.

The interface of a power domain consists of the union of the upper boundary and the lower boundary of the power domain.

create_power_domain also defines the supply sets that are used to provide power to instances within the extent of the power domain. The **-supply** option defines a supply set handle for a supply set used in the power domain.

A domain *supply_set_handle* may be defined without an association to a *supply_set_ref*. The association can be completed separately (see <u>6.7</u>).

When both a *supply_set_handle* and a *supply_set_ref* are specified with **-supply**, the following supply set association is implied:

```
associate_supply_set supply_set_ref
    -handle domain_name.supply_set_handle
```

Three supply set handles are predefined for each power domain: primary, default_isolation, and default_retention.

The primary supply set is implicitly connected to instances and logic inferred from the instances in the power domain. However, the primary supply set shall not be implicitly connected when any of the following apply:

- a) An instance has at least one supply net explicitly or automatically connected and **set_simstate_behavior** (see <u>6.53</u>) has not been enabled.
- b) An instance has set_simstate_behavior disabled.
- c) An instance is created as a result of a UPF command, e.g., isolation cells, level-shifters, power switches, and retention registers.

Implicit connections imply simulation semantics as specified in 4.6.2.

The **default_isolation** supply set is the default supply for any isolation cell inserted into this domain if no isolation supply is specified in the **set_isolation** command (see 6.41). The applicable **default_isolation** supply is based upon the domain in which the isolation cell is inserted, not the domain for which the isolation strategy is defined.

The **default_retention** supply set is the default supply for any retention cell inserted into this domain if no retention supply is specified in the **set_retention** command (see 6.49).

Within a power domain, the predefined supply sets **primary**, **default_isolation**, and **default_retention** are available for use by implementation tools as required to power instances in the extent of the domain, isolation cells placed in the domain, and retention cells placed in the domain, respectively. Supply sets identified by command options of **set_isolation** (see <u>6.41</u>), **set_level_shifter** (see <u>6.43</u>), **set_repeater** (see <u>6.48</u>), and **set_retention** (see <u>6.49</u>) are also available to power isolation, level-shifter, repeater, and retention cells, respectively, inserted into the domain. Collectively, the predefined supply set handles of a power domain and the supply sets identified by options of strategies associated with the domain are referred to as the *locally available supplies* of that domain.

The **-available_supplies** option specifies whether any additional supplies are also available for use, and if so, which supplies are available. If **-available_supplies** does not appear, all supply sets and supply set handles defined in or above the scope of the power domain are available for use by tools to power cells inserted into the power domain. If **-available_supplies** appears with an empty string argument, only the locally available supplies are available for use by tools to power domain. If **-available_supplies** appears with a non-empty string, the string shall be a list of the names of additional supply sets or supply set handles defined at or above the scope of the power domain that are also available for use by tools to power cells inserted into the power cells inserted into the power domain are available.

Any restrictions on the availability of supply sets or supply set handles for use by tools to power cells inserted into a given domain have no effect on the legality of referencing such supply sets or supply set handles in UPF commands to associate supply sets with supply set handles or to connect supply set functions explicitly, implicitly, or automatically to supply pins of an instance.

-define_func_type specifies the mapping from functions of the domain's primary supply set to *pg_type* attribute values in the *pg_type_list*. This mapping determines the automatic connection semantics used to connect the domain's primary supply to instances within the extent of the domain.

-update may be used to add elements and supplies to a previously created domain. It shall be an error if -update is used during the initial creation of *domain name*.

It shall be an error

- if an implementation tool encounters a -simulation_only power domain.
- for any instance in the descendant subtree of an atomic power domain to be included in the extent of another power domain, unless that instance name is, or is in the descendant subtree of, an instance whose name appears in the *exclude list*.
- to remove an element from an atomic power domain.
- to specify **-atomic** with **-update**.
- to specify -elements or -exclude_elements with -update for an atomic power domain.

Syntax example:

```
create_power_domain PD1 -elements {top/U1}
-supply {primary}
-supply {default_isolation}
-supply {default_retention}
-supply {mem_array ss.mem}
create power domain PD2 -elements {.}
```

6.18 create_power_switch

Purpose	Define a power switch		
Syntax	<pre>create_power_switch sw -output_supply_port {put {-input_supply_port {put {-control_port {port_na} {-on_state {state_name if [-off_state {state_name if [-supply_set supply_set_ [-on_partial_state {state [-ack_port {port_name if [-ack_delay {port_name [-error_state {state_name] [-domain domain_name] [-instance {{instance_name] [-update]</pre>	<pre>vitch_name ort_name [supply_net_name]} ort_name [supply_net_name]}* me [net_name]}* input_supply_port {boolean_expression}} {boolean_expression}]* ref] e_name input_supply_port {boolean_expression}}]* net_name [logic_value]}]* delay}]* me {boolean_expression}}]* ume}*}]</pre>	
Arguments	<pre>switch_name -output_supply_port {port_name [supply_net_name]}</pre>	The name of the switch instance to create; this shall be a simple name. The output supply port of the switch and, optionally, the net where this port connects. <i>net_name</i> is a rooted name of a supply net or supply port. It shall be an error if the <i>net_name</i> is not defined in the current scope.	

	<pre>-input_supply_port {port_name [supply_net_name]}</pre>	An input supply port of the switch and, optionally, the net where this port is connected. <i>net_name</i> is a rooted name of a supply net or supply port. It shall be an error if the <i>net_name</i> is not defined in the current scope.	
	<pre>-control_port {port_name [net_name]}</pre>	A control port on the switch and, optionally, the net where this control port connects. <i>net_name</i> is a rooted name of a logic net or logic port. It shall be an error if the <i>net_name</i> is not defined in the current scope.	
	<pre>-on_state {state_name input_supply_port {boolean_expression}}</pre>	A named on state, the <i>input_supply_port</i> for which this is defined, and its corresponding Boolean expression.	
	<pre>-off_state {state_name {boolean_expression}}</pre>	A named off state and its corresponding Boolean expression.	
	-supply_set supply_set_ref	A supply set associated with the switch. <i>supply_set_ref</i> is a rooted name of a supply set or a supply set handle. It shall be an error if the <i>supply_set_ref</i> is not defined in the current scope.	
Arguments	-on_partial_state {state_name input_supply_port {boolean_expression}}	A named partial-on state, the <i>input_supply_port</i> for which this is defined, and its corresponding Boolean expression.	
	<pre>-ack_port {port_name net_name [logic_value]}</pre>	The acknowledge port on the switch and the logic net to which this port connects. A logic value can also be specified. <i>net_name</i> is a rooted name of a logic net or logic port. It shall be an error if the <i>net_name</i> is not defined in the current scope. If a null string is used as the <i>net_name</i> for -ack_port , the port and its logic value are defined, but the port itself is unconnected.	
	<pre>-ack_delay {port_name delay}</pre>	The acknowledge delay for a given acknowledge port.	
	-error_state {state_name {boolean_expression}}}	A named error state and its corresponding Boolean expression.	
	-domain domain_name	If specified, the scope of the domain is the scope in which the switch instance is created.	
	<pre>-instance {{instance_name}*}</pre>	The hierarchical name of a technology leaf-cell instance that implements all or part of the specified switch. Instance names are the hierarchical names of the switch instances.	R
	-update	Use -update to allow the addition of -instance .	R
Return value	Return an empty string if	successful or raise a TCL_ERROR if not.	

The **create_power_switch** command defines an abstract model of a power switch. An implementation may use detailed power-switching structures that involve multiple, distributed power switches in place of a single abstract power switch.

Power-switch port names and port state names are defined in the scope of the switch instance and, therefore, can be referenced with a hierarchical name in the same way that any other instance ports can be referenced. For example, the command

```
create_power_switch PS1
   -output_supply_port {outp}
   -input_supply_port {inp}
...
```

creates an instance PS1 in the current scope and creates supply ports outp and inp within the PS1 instance. The switch supply ports can then be referred to as PS1/inp and PS1/outp.

The abstract power-switch model has one or more input supply ports and one output supply port. Each of the input supplies may contribute to the output supply as determined by control expressions. Each input supply port is effectively gated by one or more control expressions defined by *on_state* or *on_partial_state* expressions. An *on_state* expression specifies when a given input supply contributes to the output without limiting current. An *on_partial_state* expression specifies when a given input supply contributes to the output in a current-limited manner. Each input supply may have multiple *on_state* and/or *on_partial_state* expressions.

The abstract power-switch model may also have one or more error_state expressions defined. Any *error_state* expressions defined for a given power switch represent control input conditions that are illegal for that switch.

The abstract power-switch model may also have a single off_state expression defined. The off_state expression represents the condition under which no on_state or on_partial_state expression is True. If not specified explicitly, the off_state expression defaults to the complement of the conjunction of all the on_state, on_partial_state, and error_state expressions defined for the power switch. It shall be an error if the off_state expression is explicitly defined and it evaluates to True when an on_state or on_partial_state expression also evaluates to True.

A contributing input supply port is one that has an *on_state* expression or *on_partial_state* expression that evaluates to *True* at a given time. The contributed value of a contributing input supply port is the value of the supply source connected to that input supply port. The degraded value of a contributing input supply port is the contributed value, except that if the contributed value's net state is **FULL_ON**, the degraded value's net state is **PARTIAL_ON**.

The value of the output supply port of a power switch is determined as follows. At any given time:

- a) The output supply takes on the value {UNDETERMINED, unspecified} if
 - 1) any *error_state* condition is *True*, or
 - 2) an explicit off state condition and any on state or on partial state condition are both True, or
 - 3) any input supply port's contributed value has a net state of UNDETERMINED, or
 - 4) any two input supply ports' contributed values have different voltage values.
- b) Otherwise, the switch output takes on the contributed value of any contributing input supply port whose net state is **FULL_ON**, if there is one.
- c) Otherwise, the switch output takes on the degraded value of any contributing input supply port whose net state is **PARTIAL_ON**, if there is one.
- d) Otherwise, the switch takes on the value {**OFF**, unspecified}.

An anonymous root supply driver originates the state of the output supply port when the state of the output supply port is explicitly set to **UNDETERMINED** or **OFF** in the preceding algorithm.

If an **-ack_port** argument is specified, an acknowledge value is driven onto the specified *port_name delay* time units after the switch output transitions to a **FULL_ON** state and the inverse acknowledge value is driven onto the specified *port_name delay* time units after the switch output transitions to an **OFF** state.

If the supply set of the power switch is in a power state with a **NORMAL** simstate, then the acknowledge value is a logic 0 or logic 1. If a *logic_value* is specified for **-ack_port**, that logic value shall be used as the acknowledge value for a transition to **FULL_ON**, and its negation is used as the acknowledge value for a transition to **OFF**; otherwise the acknowledge value defaults to logic 1 for a transition to **FULL_ON** and logic 0 for a transition to **OFF**. If **-ack_delay** is specified, the delay may be specified as a time unit, or it may be specified as a natural integer, in which case the time unit shall be the same as the simulation precision; otherwise, the delay defaults to 0.

If **-supply_set** is specified for a switch, it powers logic or timing control circuitry within the switch and powers any specified **-ack_ports**. When the supply set simstate is anything other than **NORMAL**, the state of the output supply port of a switch is **UNDETERMINED** and the acknowledge ports are corrupted. If a supply set is not associated with a switch, it shall be an error if any acknowledge ports are specified.

-instance specifies that the power-switch functionality exists in the HDL design and *instance_name* denotes the instance providing part or all of this functionality. If -instance is specified, and a list of instances is given, then the switch may be implemented as multiple switches, in which case the multiple instances may have characteristics different from those specified by the **create_power_switch** command, particularly with regard to input and output supply connections.

An *instance_name* is a hierarchical name rooted in the current scope. If an empty string appears in an *instance_name*, this indicates that an instance was created and then optimized away. Such an instance should not be reinferred or reimplemented by subsequent tool runs.

Updating **-instance** adds the new instance names to the existing instance list. **-update** adds information to the base command executed in the same scope in which the object exists or is to be created.

The following also apply:

- Any name in a *boolean* expression shall refer to a control port of the switch.
- All states not covered by the on, on_partial, off, and error states are anonymous error states.
- If the implementation of a switch can not be inferred, map_power_switch (see <u>6.32</u>) can be used to specify it.
- If *net_name* is not specified for any of the switch's port definitions, **connect_logic_net** (see <u>6.10</u>) or **connect_supply_net** (see <u>6.11</u>) can be used to create the port connections.
- Each state name shall be unique for a particular switch.
- Any port names specified in this command are user defined (e.g., input supply).

NOTE 1—create_power_switch can be used to define an abstract power switch that implementation tools may expand into multiple switches. create_power_switch can also be used to specify the need for a specific switch that can then be mapped to a specific switch implementation using **map_power_switch**. It is not meant to be used as a single definition representing multiple physical switches to be mapped with **map_power_switch**.

NOTE 2—**create_power_switch** provides relatively simple, general abstract functionality. HDLs can be used to model switch functionality that cannot be captured with **create_power_switch**.

Power-switch examples

Example 1—Simple switch

This switch model has a single supply input and a single control input. The switch is either on or off, based on the control input value. Since net names are not specified for each port, **connect_supply_net** (see <u>6.11</u>) can be used to connect a net to each port.

```
create_power_switch simple_switch
-output_supply_port {vout}
-input_supply_port {vin}
```

```
-control_port {ss_ctrl}
-on_state {ss_on vin { ss_ctrl }}
-off state {ss off { ! ss ctrl }}
```

The following is a variant of the simple switch in which the nets associated with the ports are defined as part of the **create_power_switch** command (see 6.18).

```
create_power_switch simple_switch2
-output_supply_port {vout VDD_SW}
-input_supply_port {vin VDD}
-control_port {ss_ctrl sw_ena}
-on_state {ss_on vin { ss_ctrl }}
-off state {ss_off { ! ss_ctrl }}
```

Example 2-Two-stage switch

This switch model represents a switch that turns on in two stages. The switch has one supply input and two control inputs. One control input represents the enable for the first stage; the other represents the control for the second stage. When only the first control is on, the switch output is in a partial on state; when the second is on, the switch output is in a fully on state. The switch is off if neither control input is on.

```
create_power_switch two_stage_switch
-output_supply_port {vout}
-input_supply_port {vin}
-control_port {trickle_ctrl}
-control_port {main_ctrl}
-on_partial_state {ts_ton vin { trickle_ctrl }}
-on _state {ts_mon vin { main_ctrl }}
-off_state {ts_off { ! trickle_ctrl && ! main_ctrl }}
```

The following is a variant of the two-stage switch model in which an **-ack_port** signals completion of the switch turning on. The time required for the switch to turn on is modeled by the **-ack_delay**. Since an **-ack_port** is involved, the command needs to include specification of the supply set that powers the logic driving the ack signal. The ack signal is defined separately. In this model, as in the preceding simple switch variant, the supply and control ports are associated with corresponding nets, so they do not need to be connected in a separate step.

```
create_power_switch two_stage_switch2
-output_supply_port {vout VDD_SW}
-input_supply_port {vin VDD}
-control_port {trickle_ctrl t_ena}
-control_port {main_ctrl m_ena}
-on_partial_state {ts_ton vin { trickle_ctrl }}
-on_state {ts_mon vin { main_ctrl }}
-off_state {ts_off { ! trickle_ctrl && ! main_ctrl }}
-ack_port {ts_ack 1}
-ack_delay {ts_ack 100ns}
-supply_set ss_aon
```

Example 3-Muxed switch

This switch model represents a mux that determines which of two different input supplies is connected to the output supply port at any given time. The two input supplies can be driven by different root supply drivers and may have different state/voltage values. One control input determines which of the two input supplies is selected; the other control input gates the selected supply to the output supply.

```
create_power_switch muxed_switch
-output_supply_port {vout}
-input_supply_port {vin0}
-input_supply_port {vin1}
-control_port {ms_sel}
-control_port {ms_ctrl}
-on_state {ms_on0 vin0 { ms_ctrl && ! ms_sel }}
-on_state {ms_on1 vin1 { ms_ctrl && ms_sel }}
-off_state {ms_off { ! ms_ctrl }}
```

The following is a variant of the muxed switch in which there are two independent selection control inputs, and an error state is defined to ensure mutual exclusion.

```
create_power_switch muxed_switch2
-output_supply_port {vout}
-input_supply_port {vin0}
-input_supply_port {vin1}
-control_port {ms_sel0}
-control_port {ms_sel1}
-control_port {ms_ctrl}
-on_state {ms_on0 vin0 { ms_ctrl && ms_sel0 }}
-on_state {ms_on1 vin1 { ms_ctrl && ms_sel1 }}
-off_state {ms_off { ! ms_ctrl }}
-error_state {conflict { ms_sel0 && ms_sel1 }}
```

Example 4-Overlapping muxed switch

This switch model represents a supply mixer that allows a smooth transition between two different supplies. Like the muxed switch, it has two supply inputs and both selecting and gating control inputs, but in this case it can select both input supplies at the same time. The input supplies may have different states, and may even be driven by different root supply drivers, provided that their voltages are the same when both inputs are enabled (in an on state or on partial state).

```
create_power_switch overlapping_muxed_switch
-output_supply_port {vout}
-input_supply_port {vin0}
-input_supply_port {vin1}
-control_port {oms_sel0}
-control_port {oms_sel1}
-control_port {oms_ctrl}
-on_state {oms_on0 vin0 { oms_ctrl && oms_sel0 }}
-on_state {oms_on1 vin1 { oms_ctrl && oms_sel1 }}
-off_state {oms_off { !oms_ctrl || { !oms_sel0 && !oms_sel1 } }}
```

6.19 create_pst [legacy]

Purpose	Create a power state table (PST)	
Syntax	create_pst table_name -supplies supply_list	
	table_name	The PST name. <i>table_name</i> is a simple name in the current scope.
Arguments	-supplies supply_list	The list of supply nets or ports to include in each power state of the design. The supplies are listed as rooted names in the current scope.
Return value	Return the name of the created PST or raise a TCL_ERROR if the PST is not created.	

This is a legacy command; see also <u>6.1</u> and <u>Annex D</u>.

The **create_pst** command defines a PST name and a set of supply nets for use in **add_pst_state** commands (see <u>6.5</u>). The PST *table name* is defined in the namespace of the current scope.

A PST is used for implementation—specifically for synthesis, analysis, and optimization. It defines the legal combinations of states, i.e., those combinations of states that can exist at the same time during operation of the design.

create_pst can only be used with **add_pst_state** (and vice versa). This combination and use of **add_power_state** (see <u>6.4</u>) are two methods for specifying power state information. Power state specifications and default state definitions form an exhaustive specification of all of the legal power states of the system.

It shall be an error if

- *table_name* conflicts with any name declared in the namespace of the current scope.
- a specified supply net or supply port specified in *supply list* does not exist.

Syntax example:

```
create_pst MyPowerStateTable -supplies {PN1 PN2 SOC/OTC/PN3}
```

6.20 create_supply_net

	Purpose	Create a supply net	
	Syntax	<pre>create_supply_net net_name [-domain domain_name][-reuse] [-resolve <unresolved one_hot="" parallel="" parallel_one_hot="" ="">]</unresolved></pre>	
	Arguments	net_name	A simple name.
		-domain domain_name	The domain in whose scope the supply net is to be created.
		-reuse	Extend availability of a supply net previously defined for another domain into this domain.

Arguments	-resolve <unresolved <br="">one_hot parallel parallel_one_hot></unresolved>	A resolution mechanism that determines the state and voltage of the supply net when the net has multiple supply sources (see $6.20.2$). If no option is specified, the behavior for resolution is the same as for unresolved .
Return value	Return an empty string if successful or raise a TCL_ERROR if not.	

The **create_supply_net** command creates a supply net. If **-domain** is not specified, the supply net is created in the current scope, and the supply net is available for use by tools to power cells in any domain created at or below this scope. The net is propagated through implicitly created ports and nets throughout the logic hierarchy in the descendant tree of the scope in which the net is created as required by implicit and automatic connections of supply sets (see <u>6.17</u>).

If **-domain** is specified, the supply net is created in the scope of that domain, and the supply net is available for use by tools to power cells only in the extent of the domain.

If **-reuse** is specified, a supply net with this name needs to have been created by a previously executed command, and this existing supply net is made available for use in another domain by the **-domain** option. In this case:

- a) -domain shall also be specified on both this and the creating command;
- b) -resolve shall not conflict with that of the creating command.

The following also apply:

- It shall be an error if *domain name* is not the name of a previously created power domain.
- When -reuse is specified, it shall be an error if *net_name* is not defined for another power domain in the same scope by another create_supply_net command.
- When the parameter for **-resolve** is **unresolved**, the supply net shall have only one source (see <u>6.20.1</u>). For all other parameters to **-resolve**, the requirements on the drivers and sources of the net are as defined in <u>6.20.2</u>.

NOTE—Use set_scope (see 6.52) to change the scope prior to calling this command to set the current scope to the correct scope for the net.

Syntax example:

```
create_supply_net local_vdd_3
    -resolve one_hot
```

6.20.1 Supply net resolution

Supply nets are often connected to the output of a single switch. However, certain applications, such as onchip voltage scaling, may require the outputs of multiple switches or other supply drivers to be connected to the same supply net (either directly or via supply port connections). In these cases, a resolution mechanism is needed to determine the state and voltage of the supply net from the state and voltage values supplied by each of the supply drivers to which the net is connected.

A supply net that specifies an **unresolved** resolution cannot be connected to more than one supply source.

6.20.2 Resolutions methods

The semantics of each possible resolution method are as follows:

a) unresolved

The supply net shall be connected to at most one supply source. This is the default.

b) one_hot

Multiple supply sources, each having a unique driver, may be connected to the supply net.

A supply net with **one_hot** resolution has a deterministic state only when no more than one source drives the net at any particular point in time. If at any point in time more than one supply source driving the net is anything other than **OFF**, the state of the supply net shall be **UNDETERMINED**, the voltage value of the supply net shall be unspecified, and implementations may issue a warning or an error.

- 1) If all supply sources are **OFF**, the state of the supply net shall be **OFF**, and the voltage value of the supply net shall be unspecified.
- 2) If only one supply source is **FULL_ON** and all other sources are **OFF**, the state of the supply net shall be **FULL_ON**, and the voltage value of the corresponding source shall be assigned to the supply net.
- 3) If only one supply source is **PARTIAL_ON** and all other sources are **OFF**, the state of the supply net shall be **PARTIAL_ON** and the voltage value of the corresponding source shall be assigned to the supply net.
- 4) If any source is **UNDETERMINED**, the state of the supply net shall be **UNDETERMINED**, and the voltage value of the supply net shall be unspecified.

c) parallel

Multiple supply sources, sharing a common root supply driver, may be connected to the supply net.

The **parallel** resolution allows more than one potentially conducting path to the same root supply driver, as if the switches had been connected in parallel. It shall be an error if any of these potentially conducting paths can be traced to more than one root supply driver.

- 1) If all of the supply sources are **FULL_ON**, then the supply net state is **FULL_ON** and the voltage value is the value of the root supply driver.
- 2) If all the supply sources driving the supply net are **OFF**, the state of the supply net shall be **OFF** and the voltage is unspecified.
- 3) If any of the sources is UNDETERMINED, the resolution is UNDETERMINED; otherwise,
 - i) If there is at least one **PARTIAL_ON** source, the supply net shall be **PARTIAL_ON** and the voltage value is the value of the root supply driver.
 - ii) If there is at least one source that is **OFF** and at least one that is **FULL_ON** or **PARTIAL_ON**, the supply net shall be **PARTIAL_ON** and the voltage value is the value of the root supply driver. The voltage value of the **PARTIAL_ON** supply net shall be the voltage value of the root supply driver.

d) parallel_one_hot

Multiple supply sources may be connected to the supply net. A source may share a common root supply driver with one or more other sources. At most one root supply driver shall be **FULL_ON** at any particular time with all sources sharing that driver resolved using parallel resolution.

The **parallel_one_hot** resolution allows resolution of a supply net that has multiple root supply drivers where each driver may have more than one path through supply sources to the supply net. Each unique root supply driver is identified and **one_hot** resolution shall be applied to the drivers, then **parallel** resolution shall be applied to each supply source connecting the **one_hot** root supply driver to the supply net.

6.20.3 Supply nets defined in HDL

The declaration of any VHDL signal or SystemVerilog wire or reg as a supply_net_type from the package UPF (see <u>Annex B</u>) is equivalent to calling **create_supply_net** for every instance of that declaration, where the *net_name* is the name of the VHDL signal or SystemVerilog wire or reg, and the scope is the instance.

6.21 create_supply_port

Purpose	Create a supply port on a instance	
Syntax	<pre>create_supply_port port_name [-domain domain_name] [-direction <in inout="" out="" ="">]</in></pre>	
	port_name	A simple name.
Arguments	-domain domain_name	The domain where this port defines a supply net connection point.
	-direction <in out="" ="" <br="">inout></in>	The direction of the port. The default is in .
Return value	Return an empty string if successful or raise a TCL_ERROR if not.	

The **create_supply_port** command defines a supply port at the scope of the power domain when **-domain** is specified or at the current scope if **-domain** is not specified.

-direction defines how state information is propagated through the supply network as it is connected to the port. If the port is an input port, the state information of the external supply net (see 6.20) connected to the port shall be propagated into the instance. Likewise, for an output port, the state information of the internal supply net connected to the port shall be propagated outside the instance.

Supply ports connected to a net shall be **inout** for supply nets that have both loads and sources within that module. Supply ports are loads, sources, or both, as follows:

- a) The LowConn of an input port is a source.
- b) The HighConn of an input port is a sink.
- c) The LowConn of an output port is a sink.
- d) The HighConn of an output port is a source.
- e) The LowConn of an inout port is both a source and a sink.
- f) The HighConn of an inout port is both a source and a sink.

Supply ports may be defined in HDL. If a VHDL or SystemVerilog port is declared as a supply_net_type from the package UPF (see <u>Annex B</u>), this is equivalent to calling **create_supply_port** for every instance of that declaration, where the *port_name* is the name of the VHDL or SystemVerilog port, and the scope is the instance.

Syntax example:

```
create_supply_port VN1
    -direction inout
```

6.22 create_supply_set

Purpose	Create or update a supply set, or update a supply set handle		
Syntax	create_supply_set_set_name [-function {func_name net_name}]* [-reference_gnd supply_net_name] [-update]		
	set_name	The simple name of the supply set or a supply set handle.	
	-function {func_name net_name}	The -function option defines the function (<i>func_name</i>) a supply net provides for this supply set. <i>net_name</i> is a rooted name of a supply net or supply port or a supply net handle. It shall be an error if the net_name is not defined in the current scope.	R
Arguments	<pre>-reference_gnd supply_net_name</pre>	The -reference_gnd option defines the rooted name of a <i>supply_net</i> that serves as the reference ground for the supply set. A supply net handle may be used. Default : if not specified, the voltages in this supply set shall be evaluated with no offset from the assumed default reference supply.	R
	-update	Use -update if the <i>set_name</i> has already been defined.	R
Return value	Return an empty string i	f successful or raise a TCL_ERROR if not.	

create_supply_set creates the supply set name within the current scope in the UPF name space. The reference ground can be specified in any invocation of this command. This command defines a *supply set* as a collection of supply nets each of which serve a specific function for the set.

-update is used to signify that this create_supply_set call refers to a supply set that was previously defined using create_supply_set, or to a supply set handle that was previously defined implicitly or explicitly using create_power_domain (see <u>6.17</u>). Referencing a previously created supply set or supply set handle without the -update argument shall be an error. Using the -update argument for a supply set that has not been previously defined shall be an error. Specifying a supply set handle that has not been previously defined shall be an error.

When **-function** is specified, *func_name* shall be one of the following: **power**, **ground**, **nwell**, **pwell**, **deepnwell**, and **deeppwell**. The **-function** option associates the specified *func_name* of this supply set with the specified *supply_net_name*. If the same *func_name* is associated with two different supply nets, it shall be an error if those supply nets are not the same. The *supply_net_name* may be a reference to a supply net in the descendant hierarchy of the current scope using a supply net handle (see <u>6.22.1</u>).

When **-reference_gnd** is specified, *supply_net_name* is the name of a supply net that serves as the reference ground for the supply set. The voltage value for each supply net in the supply set is interpreted in reference to this supply net. If this parameter is not specified, the voltages shall be evaluated with no offset or scaling. If **-reference_gnd** has previously had a *supply_net_name* specified, then it shall be an error if this *supply_net_name* and the *supply_net_name* previously specified as reference ground are not equivalent nets.

Syntax example:

```
create_supply_set relative_always_on_ss
-function {power vdd}
-function {ground vss}
```

```
create_supply_set relative_always_on_ss -update
    -reference_gnd {earth_ground}
    create_supply_set PD1.primary -update
        -function {nwell bias}
```

6.22.1 Referencing supply set functions

The supply set function may also be referenced using a supply net handle as a member of the supply set (whether or not a supply net has been associated with the function name), as follows:

supply_set_ref.function

6.22.2 Implicit supply net

If no supply net is associated with a supply set's function and that function is used in the design, an implicit supply net with an anonymous name shall be created for use in verification and analysis. When the UPF specification is used for implementation, a supply net shall not be implicitly created for a supply set function that has no associated supply net. A tool may issue a warning or an error if a supply set's function does not have an explicit supply net association.

6.23 create_upf2hdl_vct

Purpose	Define VCT that can be used in converting UPF <pre>supply_net_type</pre> values into HDL logic values	
Syntax	<pre>create_upf2hdl_vct vct_name -hdl_type {<vhdl sv="" =""> [typename]} -table {{from_value to_value}*}</vhdl></pre>	
	vct_name	The VCT name.
Arguments	-hdl_type { <vhdl sv="" =""> [typename]}</vhdl>	The HDL type for which the value conversions are defined.
	<pre>-table {{from_value to_value}*}</pre>	A list of UPF state type values to map to the values of the HDL type.
Return value	Return an empty string if successful or raise a TCL_ERROR if not.	

The **create_upf2hdl_vct** command defines a value conversion table (VCT) for the supply_net_type.state value (see <u>Annex B</u>) when that value is propagated from a UPF supply net into a logic port defined in an HDL. It provides a 1:1 conversion for each possible combination of the partially on and on/off states. **create_upf2hdl_vct** does not check that the values are compatible with any HDL port type.

vct_name provides a name for the value conversion table for later use with the **connect_supply_net** command (see <u>6.11</u>). The predefined VCTs are shown in <u>Annex F</u>.

-hdl_type specifies the HDL type for which the value conversions are defined. This information allows a tool to provide completeness and compatibility checks. If the *typename* is not one of the language's predefined types or one of the types specified in the next paragraph, then it shall be of the form *library*.pkg.type.

The following HDL types shall be the minimum set of types supported. An implementation tool may support additional HDL types.

- a) VHDL
 - 1) Bit, std [u]logic, Boolean
 - 2) Subtypes of std_[u]logic
- b) SystemVerilog

reg/wire, Bit, Logic

-table defines the 1:1 conversions from UPF supply net states to an HDL logic value. The values shall be consistent with the HDL type values. For example:

- When converting to SystemVerilog *logic type*, the set of legal values is 0, 1, X, and Z.
- When converting to SystemVerilog or VHDL bit, the legal values are 0 or 1.
- When converting to VHDL std [u]logic, the legal values are U, X, O, 1, Z, W, L, H, and -.

The conversion values have no semantic meaning in UPF. The meaning of the conversion value is relevant to the HDL model to which the supply net is connected.

Syntax examples:

```
create_upf2hdl_vct upf2vlog_vdd
  -hdl_type {sv}
  -table {{OFF X} {FULL_ON 1} {PARTIAL_ON 0}}
create_upf2hdl_vct upf2vhdl_vss
  -hdl_type {vhdl std_logic}
  -table {{OFF `X'} {FULL_ON `1'} {PARTIAL_ON `H'}}
```

6.24 describe_state_transition

Purpose	Describe a state transition's legality	
Syntax	<pre>describe_state_transition transition_name -object object_name [-from from_list -to to_list] [-paired {{from_state to_state}*}] [-legal -illegal]</pre>	
Arguments	transition_name	Simple name.
	<pre>-object object_name</pre>	Simple name of a power domain or supply set.
	-from from_list -to to_list	<i>from_list</i> is an unordered list of power state names active before a state transition. <i>to_list</i> is an unordered list of power state names active after a state transition.
	<pre>-paired {{from_state to_state}*}</pre>	A list of from-state name and to-state name pairs.
	-legal -illegal	Define the state transition as legal or illegal, the default is -legal .
Return value	Return an empty string if successful or raise a TCL_ERROR if not.	

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

The option **-from** and **-to** specify many-to-many transitions. The option **-paired** specifies one or more one-to-one transitions. At least one of these two choices shall be specified.

If an empty list is specified in either the **-from** or **-to** *list*, it shall be expanded to all named power states for the specified *object name*.

Verification tools shall emit an error when an illegal state transition occurs.

It shall be an error if the state name in a *list* does not refer to a power state of the specified supply state or power domain (see 6.4).

Syntax example:

```
describe_state_transition turn_on -object PdA -from {SLEEP_MODE}
   -to {HIGH_SPEED_MODE} -illegal
```

6.25 end_power_model

Purpose	Terminate the definition of a power model
Syntax	end_power_model
Arguments	N/A
Return value	Return a 1 if successful or raise a TCL_ERROR if not.

The **begin_power_model** (see <u>6.8</u>) and **end_power_model** commands define a power model containing other UPF commands. A power model is used to define the power intent of a hard IP and shall be used in conjunction with one or more model representations. A power model defined with **begin_power_model** is terminated by the first subsequent occurrence of **end_power_model** in the same UPF file.

6.26 find_objects

Purpose	Find logic hierarchy objects within a scope	
Syntax	<pre>find_objects scope -pattern search_pattern [-object_type <model inst="" net="" port="" process="" ="">] [-direction <in inout="" out="" ="">] [-transitive [<true false="" ="">]] [-regexp -exact] [-ignore_case] [-non_leaf -leaf_only]</true></in></model></pre>	
Arguments	scope	The search is restricted to the specified scope.
	-pattern search_pattern	The string used for searching. By default, <i>search_pattern</i> is treated as an Tcl glob expression.
	-object_type <model <br="">inst port net process></model>	Limits the objects returned. By default, instances, named processes, ports, and nets are returned; this can be restricted by specifying a specific -object_type . inst does not return named processes.
	-direction <in out="" ="" <br="">inout></in>	If -object_type is port , then -direction can be used to restrict the directions of the returned ports.
	-transitive [<true <br="">FALSE>]</true>	If -transitive is not specified at all, the default is -transitive FALSE . If -transitive is specified without a value, the default value is TRUE .
	-regexp -exact	-regexp enables support for regular expression in the specified <i>search_pattern</i> . -exact disallows wildcard expansion on the specified <i>search_pattern</i> . If neither -regexp or -exact are specified, then <i>search_pattern</i> is interpreted as a Tel glob expression.
	-ignore_case	Performs case-insensitive searches. By default, all matches are case sensitive.
	-non_leaf -leaf_only	If -non_leaf is specified, only non-leaf instances (instances that have children) are returned; if -leaf_only is specified, only leaf-level instances (instances without children) are returned. By default, both leaf and non-leaf instances are returned.
Return value	Returns a list of names (relative to the current scope) of objects that match the search criteria; when nothing is found that matches the search criteria, a <i>null string</i> is returned. The list contains just the object names, without any indication of object type. The list may contain names of more than one type of object.	

The **find_objects** command searches for instances, nets, ports, or processes that are defined in the logic hierarchy. If **-object_type model** is specified, **find_objects** searches for instances of any model whose model name matches the search_pattern. If **-object_type** is specified with any other value, **find_objects** searches the logic hierarchy for the specified objects whose name matches the *search_pattern*.

By default, or if **-transitive FALSE** is specified explicitly, **find_objects** searches only the current scope of the logic hierarchy. If **-transitive TRUE** is specified, **find_objects** searches the current scope and the entire dependant subtree. If **-transitive** is specified without an argument, it is equivalent to specifying **-transitive TRUE**.

NOTE—To find UPF objects, such as isolation logic or retention elements, use the corresponding query_* commands (see <u>Annex C</u>).

The **-non_leaf** and **-leaf_only** options can be interpreted differently between tools, depending upon the library source. For example, in simulation, an IP block may be represented as a non-leaf hierarchical behavioral model, whereas in implementation, the same IP block may be represented as a black box leaf cell. A module may be tagged as a leaf cell by using **set design attributes** (see <u>6.37</u>).

The following conditions also apply:

- The specified *scope* cannot start with . . or /, i.e., **find_objects** shall be referenced from the current scope, and reside in the current scope or below it.
- If *scope* is specified as . (a dot), the current scope is used as the root of the search.
- All elements returned are referenced to the current scope.
- It shall be an error if *scope* is neither the current scope nor is defined in the current scope. The specified scope may reference a generate block as the root of the search.
- While find_objects commands are executed and their results are used; the command itself is not saved. However, this does not prohibit the use of find objects in output UPF.

Syntax examples:

```
find_objects A/B/D -pattern *BW1*
    -object_type inst
    -transitive TRUE
```

6.26.1 Pattern matching and wildcarding

To improve usability and allow multiple objects (instances, ports, etc.) to be easily specified without onerous verbosity, pattern matching (wildcarding) is allowed (only) in **find_objects** and **query_upf** (see <u>C.1</u>). Pattern matching is supported using the Tcl glob style, matching against the symbols in the scope rather than filenames. For glob-style wildcarding, the following special operators are supported:

? matches any single character.

* matches any sequence of zero or more characters.

[*chars*] matches any single character in *chars*. If *chars* contains a sequence of the form a-b, any character between a and b (inclusive) shall match.

x matches the character x.

 $\{a,b,c\}$ Matches any string that is matched by any of the patterns a, b, or c.

Tcl regular expression matching is described in the Tcl documentation for re syntax (see [B5]).

6.26.2 Wildcarding examples

Table 5 shows the pattern match for each of the following examples of find_objects.

```
find_objects top -pattern a
find_objects top -pattern {bc[0-3]}
find_objects top -pattern e*
find_objects top -pattern d?f
find_objects top -pattern {g\[0\]}
```

NOTE 1—The use of the Tcl quote semantics of "{string}" in the example illustrates an effective means to pass characters that would otherwise be "special" to a Tcl interpreter.

NOTE 2—To select the four bits (0 to 3) of the bus my_bus , use the Tcl expression $\{my_bus \setminus [[0-3]]\}$.

	Only matches an instance called a in the current scope.
3]	Matches any instance called bc followed by a numerical value from 3, i.e., bc0, bc1, bc2, and bc3.
	Matches any instance starting with e, i.e., e12, eab, ef, etc.

0 to

Table 5—Pattern matches

d?fMatches any instance starting with d followed by another character and
ending in f, i.e., daf, d4f, etc.g\[0\]Matches an instance called g[0].

6.27 load_simstate_behavior

а

e*

bc[0-

Purpose	Load the simstate behavior defaults for a library	
Syntax	load_simstate_behavior lib_name -file file_list	
Arguments	lib_name	The tool specific library name for which the simstate behavior file is to be loaded.
	-file file_list	The list of files containing the set_simstate_behavior commands.
Return value	Return an empty string if successful or raise a TCL_ERROR if not.	

Loads a UPF file that only contains **set_simstate_behavior** commands and applies these to the models in the library *lib_name*.

It shall be an error if

- *lib_name* cannot be resolved.
- *file_list* does not exist.
- a model specified in *file_list* cannot be found.
- the set_simstate_behavior commands in *file list* use the -lib argument.
- *file_list* contains UPF commands other than **set_simstate_behavior**.

Syntax example:

load_simstate_behavior library1 -file simstate_file.upf

6.28 load_upf

Purpose	Set the scope to the specified instance and execute the specified UPF commands	
Syntax	load_upf_upf_file_name [-scope instance_name_list] [-version upf_version]	
Arguments	upf_file_name	The UPF file to execute.
	-scope instance_name_list	The list of scopes where the UPF commands contained in <i>upf_file_name</i> are executed.
	<pre>-version upf_version</pre>	The UPF version for which commands in this file are written. See also 6.54 .
Return value	Return an empty string if successful or raise a TCL_ERROR if not.	

The **load_upf** command sets the scope to each of the specified list of instances and executes the set of UPF commands in the file *upf_file_name*. Upon return, the current scope is restored to what it was prior to invocation. If a scope specified in *instance_name_list* is not found, further processing of remaining scopes in the *instance_name_list* is terminated and a TCL_ERROR is raised.

load_upf does not create a new name space for the loaded UPF file. The loaded UPF file is responsible for ensuring the integrity of both its own and the caller's name space as needed using existing Tcl name space management capabilities.

If **-scope** is specified, each instance name in the instance name list shall be a simple name or a hierarchical name rooted in the current scope. In this case, for the duration of the **load_upf** command, the current scope and design top instance are both set to the instance specified by the instance name and the design top module is set to the module type of that instance.

When the **load_upf** command completes, the current scope, design top instance, and design top module all revert to their previous values.

If -version upf version is specified, the command

upf_version upf_version

is implicitly executed before executing the commands in the loaded file.

Syntax example:

load upf my.upf -scope {I1/I2 I3/I2} -version 2.1

6.29 load_upf_protected

Purpose	Load a UPF file in a protected environment that prevents corruption of existing variables	
Syntax	load_upf_protected upf_file_name [-hide_globals] [-scope instance_name_list] [-version upf_version] [-params param_list]	
Arguments	upf_file_name	The UPF file to be sourced.
	-hide_globals	Save all globals before sourcing <i>upf_file_name</i> and restore them afterwards. Globals named in the <i>param_list</i> retain any modified values resulting from sourcing the file. Any globals not in the <i>param_list</i> shall be unset before <i>upf_file_name</i> is loaded. Any globals created in the sourced file, other than the ones named in <i>param_list</i> , are unset at the end of loading.
	-scope instance_name_list	The list of scopes where the UPF commands contained in <i>upf_file_name</i> are executed.
	-version upf_version	The UPF version for which commands in this file are written. See also 6.54 .
	-params param_list	 A list of variables to be made available while sourcing the file. In <i>param_list</i>, each element has one of the following formats: a) <i>param name</i> — declared as "global \$paramName". Any
		changes made to this variable are visible at the calling level once this command completes.
		b) { <i>param_name param_value</i> } — a local variable <i>param_name</i> is created and its initial value is set to <i>param_value</i> .
		The Tcl variable errorInfo shall behave as if it has been specified in this list.
Return value	Return an empty string if	successful or raise a TCL_ERROR if not.

If a scope specified in *instance_name_list* is not found, further processing of remaining scopes in the *instance_name_list* is terminated and a TCL ERROR is raised.

If **-scope** is specified, each instance name in the instance name list shall be a simple name or a hierarchical name rooted in the current scope. In this case, for the duration of the **load_upf_protected** command, the current scope and design top instance are both set to the instance specified by the instance name and the design top module is set to the module type of that instance.

When the **load_upf_protected** command completes, the current scope, design top instance, and design top module all revert to their previous values.

If -version upf version is specified, the command

upf_version upf_version

is implicitly executed before executing the commands in the loaded file.

Syntax example:

load_upf_protected my.upf -hide_globals -version 2.0
6.30 map_isolation_cell [deprecated]

This is a deprecated command; see also 6.1 and Annex D.

6.31 map_level_shifter_cell [deprecated]

This is a deprecated command; see also <u>6.1</u> and <u>Annex D</u>.

6.32 map_power_switch

Purpose	Specify which power-switt instance	tch model is to be used for the implementation of the corresponding switch	
Syntax	<pre>map_power_switch switch_name_list -lib_cells lib _cells_list [-port_map {{mapped_model_port switch_port_or_supply_net_ref}*}] [-domain domain_name]</pre>		
	switch_name_list	A list of switches [as defined by create_power_switch (see 6.18)] to map.	
	-lib_cells lib_cells_list	A list of library cells.	
Arguments	<pre>-port_map {{mapped_model_port switch_port_or_supply_ net_ref}*}</pre>	<i>mapped_model_port</i> is a port on the model being mapped. <i>switch_port_or_supply_net_ref</i> indicates a supply or logic port on a switch: an input supply port, output supply port, control port, or acknowledge port; or it references a supply net from a supply set associated with the switch. See also create_power_switch (<u>6.18</u>) or set_power_switch (<u>6.47</u>).	
Deprecated arguments	-domain domain_name	This argument is ignored and provided for syntactic backward compatibil- ity only. This is a deprecated option; see also 6.1 and Annex D	
Return value	Return an empty string if	successful or raise a TCL_ERROR if not.	

The **map_power_switch** command can be used to explicitly specify which power-switch model is to be used for the corresponding switch instance.

-lib_cells specifies the set of library cells to which an implementation can be mapped. Each cell specified in -lib_cells shall be defined by a define_power_switch_cell command (see <u>7.6</u>) or defined in the Liberty file with required attributes.

If **-port_map** is not specified, the ports of the switch instance are associated to library cell ports by matching the respective port names, this is *named association*. It shall be an error if any ports on either the switch instance or the library cell are not mapped when named association is used.

It shall be an error if switch name list is an empty list.

NOTE 1—All **map_*** commands specify the elements to be used rather than inferred through a strategy. The behavior of this manual mapping may lead to an implementation that is different from the RTL specification. Therefore, logical equivalence checking tools may not be able to verify the equivalence of the mapped element to its RTL specification.

NOTE 2—**create_power_switch** can be used to define an abstract power switch that implementation tools may expand into multiple switches. **create_power_switch** can also be used to specify the need for a specific switch that can then be mapped to a specific switch implementation using **map_power_switch**. It is not meant to be used as a single definition representing multiple physical switches to be mapped with **map_power_switch**.

Syntax example:

```
map_power_switch switch_sw1
-domain test_suite
-lib_cells {sw1}
-port_map {{inp1 vin1} {inp2 vin2} {outp vout}
        {c1 ctrl_small} {c2 ctrl_large}}
```

6.33 map_retention_cell

Purpose	Constrain implementation alternatives, or specify a functional model, for retention strategies		
Syntax	map_retention_cell retention_name_list -domain domain_name [-elements element_list] [-exclude_elements exclude_list] [-lib_cells lib_cell_list] [-lib_cell_type lib_cell_type] [-lib_model_name name -port_map {{port_name net_ref}*}]		
	retention_name_list	A list of target retention strategy names defined in <i>domain_name</i> using set_retention commands (see <u>6.49</u>).	
	-domain domain_name	The domain in which the strategies are defined.	
Arguments	-elements element_list	A list of instances, named processes, or sequential reg or signal names whose respective sequential elements shall be mapped as specified.	
	-exclude_elements exclude_list	A list of instances, named processes, or sequential reg or signal names whose respective sequential elements shall be excluded from mapping.	
	-lib_cells lib_cell_list	A list of library cell names. Each cell in the list has retention behavior and is otherwise identical to the inferred RTL behavior of the underlying sequential element.	
	-lib_cell_type lib_cell_type	The attribute of the library cells used to identify cells that have retention behavior and are otherwise identical to the inferred RTL behavior of the underlying sequential element.	
	-lib_model_name model_name -port_map {{port_name net_ref}*}	The name of the library cell or behavioral model and associated port connectivity.	
Return value	Return an empty string if	successful or raise a TCL_ERROR if not.	

The **map_retention_cell** command constrains retention strategy implementation choices and may also specify functional retention behavior for verification.

-elements identifies elements from the *effective_element_list* (see <u>5.10</u>) from a retention strategy in *retention_name_list*. If -elements is not specified, the *aggregate_element_list* for this command contains all elements from the *effective element list* of the *retention name_list*.

It shall be an error if at least one of -lib_cells, -lib_cell_type, or -lib_model_name is not specified.

If -lib_cells is specified, each cell shall be either defined by the define_retention_cell command (see 7.7) or defined in the Liberty file with required attributes;

If -lib_cells is specified, a retention cell from *lib_cell_list* shall be used; if -lib_cell_type is specified, a retention cell with the same type string specified by define_retention_cell -cell_type shall be used to implement the functionality specified by the corresponding retention strategy; if -lib_cells and -lib_cell_type are both specified, a retention cell from *lib_cell_list* that is also defined with the same type string in define_retention_cell -cell_type shall be used. Verification semantics are unchanged by the presence or absence of -lib cells or -lib cell type.

- If -lib_model_name is specified, model_name shall be used as the verification model, and supply and logic ports shall be connected as specified by -port_map options; automatic corruption and retention verification semantics do not apply to a -lib_model_name model.
- If -lib_model_name is not specified, the verification semantic is that of the inferred RTL behavior of the underlying sequential element modified by the retention behavior prescribed by the applicable set_retention strategy.

<u>Table 6</u> summarizes the semantics for combinations of **-lib_cells**, **-lib_cell_type**, and **-lib_model_name**.

-lib_cells	-lib_cell_type	-lib_model_name	Verification semantic	Implementation cell constrained to
Ν	Ν	Ν	ERROR	ERROR
N	N	Y	model_name	model_name
N	Y	Ν	RTL with retention	lib_cell_type
N	Y	Y	model_name	lib_cell_type
Y	N	Ν	RTL with retention	lib_cell_list
Y	N	Y	model_name	lib_cell_list
Y	Y	Ν	RTL with retention	A cell from <i>lib_cell_list</i> that also has <i>lib_cell_type</i>
Y	Y	Y	model_name	A cell from <i>lib_cell_list</i> that also has <i>lib_cell_type</i>

Table 6—map_retention_cell option combinations

For verification, an inferred register is assumed to have the following generic canonical interface:

- CLOCK—The signal whose rising edge triggers the register to load data.
- **DATA**—The signal whose value represents the next state of the register.
- ASYNC_LOAD—The signal that causes the register to load data when its value is one (1).
- **OUTPUT**—The signal that propagates the register output to the receivers of the register.

-port_map connects the specified *net_ref* to a *port* of the model. A *net_ref* may be one of the following:

- a) A logic net name
- b) A supply net name
- c) One of the following symbolic references
 - 1) **retention_ref**.function_name

This names a retention supply set function, where *function_name* refers to the supply net corresponding to the function it provides to the retention *ret_supply_set* (see <u>6.49</u>).

2) **primary_ref**.function_name

This names a primary supply set function, where *function_name* refers to the supply net corresponding to the function it provides to the primary supply set of the domain.

3) save_signal

- i) Refers to the save signal specified in the corresponding retention strategy.
- ii) To invert the sense of the save signal, the Verilog bit-wise negation operator ~ can be specified before the *net_ref*. The logic inferred by the negation shall be implicitly connected to the *ret_supply_set* from the corresponding set_retention command (see <u>6.49</u>).

4) restore_signal

- i) Refers to the restore signal specified in the corresponding retention strategy.
- ii) To invert the sense of the restore signal, the Verilog bit-wise negation operator ~ can be specified before the *net_ref*. The logic inferred by the negation shall be implicitly connected to the *ret_supply_set* from the corresponding **set_retention** command (see 6.49).

5) UPF_GENERIC_CLOCK

- i) Refers to the canonical **CLOCK**.
- ii) To invert the sense of the clock signal, the Verilog bit-wise negation operator ~ can be specified before the *net_ref*. The logic inferred by the negation shall be implicitly connected to the *primary supply set*.

6) UPF_GENERIC_DATA

- i) Refers to the canonical **DATA**.
- ii) To invert the sense of the data signal, the Verilog bit-wise negation operator ~ can be specified before the *net_ref*. The logic inferred by the negation shall be implicitly connected to the *primary_supply_set*.

7) UPF_GENERIC_ASYNC_LOAD

- i) Refers to the canonical **ASYNC_LOAD**.
- ii) To invert the sense of the asynchronous load signal, the Verilog bit-wise negation operator ~ can be specified before the *net_ref*. The logic inferred by the negation shall be implicitly connected to the *primary supply set*.

8) UPF_GENERIC_OUTPUT

- i) Refers to the canonical **OUTPUT**.
- ii) To invert the sense of the output signal, the Verilog bit-wise negation operator ~ can be specified before the *net_ref*. The logic inferred by the negation shall be implicitly connected to the *primary_supply_set*.

If **UPF_GENERIC_OUTPUT** is not explicitly mapped and the model has exactly one output port, that output port shall automatically be connected to the net that propagates the register output to the receivers of the register.

NOTE—All **map_*** commands specify the elements to be used rather than inferred through a strategy. The behavior of this manual mapping may lead to an implementation that is different from the RTL specification. Therefore, it may not be possible for logical equivalence checking tools to verify the equivalence of the mapped element to its RTL specification.

It shall be an error if

- retention name list is an empty list.
- *domain name* does not indicate a previously created power domain.
- A retention strategy in *retention_name_list* does not indicate a previously defined retention strategy.

- An element in *element list* is not included in the element list of a targeted retention strategy.
- Any retention strategy in *retention_name_list* does not specify signals needed to provide connection of the mapped functions.
- After completing the *port* and *net_ref* connections, any input port is unconnected, or no output port is
 connected to the net that propagates the register output to the receivers of the register.
- In implementation, none of the specified models in *lib_cell_list* implements the functionality specified by a targeted retention strategy.
- In implementation, none of the specified models having a *lib_cell_type* attribute implements the functionality specified by a targeted retention strategy.
- In implementation, none of the specified models in *lib_cell_list* that have a *lib_cell_type* attribute, when both are specified, implements the functionality specified by a targeted retention strategy.

Syntax example:

```
map retention cell {my PDA ret strat 1 my PDA ret strat 2 my PDA ret strat 3}
   -domain PowerDomainA
   -elements {foo/U1 foo/U2}
   -lib cells {RETFFIMP1 RETFFIMP2}
   -lib_cell_type FF_CKLO
   -lib model name RETFFVER -port map {
      {CP
             UPF GENERIC CLOCK}
               UPF GENERIC DATA }
      { D
             UPF GENERIC ASYNC LOAD}
      {SET
      {SAVE
             save signal}
      {RESTORE restore signal}
      {VDDC
               primary_supply_set.power}
      {VDDRET ret_supply_set.power}
      {VSS
               primary supply set.ground} }
```

6.34 merge_power_domains [deprecated]

This is a deprecated command; see also 6.1 and Annex D.

6.35 name_format

Purpose	Define the format for constructing names of implicitly created objects		
Syntax	name_format [-isolation_prefix string] [-isolation_suffix string] [-level_shift_prefix string] [-level_shift_suffix string] [- implicit_supply_suffix string] [- implicit_logic_prefix string] [- implicit_logic_suffix string]		
Arguments	-isolation_prefix string	The string prepended in front of an existing signal or port name to create a new name used during the introduction of a new isolation cell. The default value is the empty string "" or NULL.	
	-isolation_suffix string	The string appended to the end of an existing signal or port name to create a new name used during the introduction of a new isolation cell. The default value is the string _UPF_ISO.	
	<pre>-level_shift_prefix string</pre>	The string prepended in front of an existing signal or port name to create a new name used during the introduction of a new level-shifter cell. The default value is the empty string "" or NULL.	
	-level_shift_suffix string	The string appended to the end of an existing signal or port name to create a new name used during the introduction of a new level-shifter cell. The default value is the string _UPF_LS.	
	-implicit_supply_suffix string	The string appended to an existing supply net or port name to create a unique name for an implicitly created supply net or port. The default value is the string _UPF_IS.	
	-implicit_logic_prefix string	The string prepended in front of an existing logic net or port name to create a unique name for an implicitly created logic net or port. The default value is NULL.	
	-implicit_logic_suffix string	The string appended to an existing logic net or port name to create a unique name for an implicitly created logic net or port. The default value is the string _UPF_IL.	
Return value	Return an empty string if	successful or raise a TCL_ERROR if not.	

Inferred objects have names in the logic design. The name for these objects is constructed as follows:

- a) The base name of implicitly created objects is the name of the port or net being isolated or levelshifted, or the supply net, logic net, or port implicitly created to facilitate the connection of a net across hierarchy boundaries.
- b) Any specified prefix is then prepended to the base name.
- c) Any specified suffix is also appended to the base name.
- d) If multiple prefixes or suffixes apply to the same object, they shall be added in the alphabetical order of the option name, e.g., **isolation_prefix** followed by **level_shift_prefix**.

If the generated name conflicts with another previously defined name in the same name space, the generated name is further extended by an underscore (_) followed by a positive integer. The value of the integer is the smallest number that makes the name unique in its name space. An empty string ("") is a valid value for any prefix or suffix option. When the prefix and suffix are both NULL, only the underscore (_) and number string combination are used as a suffix to disambiguate the name.

Different prefixes and suffixes may be specified in multiple calls to **name_format** (using different arguments). When **name_format** is specified with no options, the name format is reset to the default values shown in the *Arguments* list.

It shall be an error to specify an affix more than once.

Syntax example:

name format -isolation prefix "MY ISO " -isolation suffix ""

A signal, MY ISO FOO, is created and connected to a new cell's output (to isolate the existing net FOO).

6.36 save_upf

Purpose	Create a UPF file of the s	tructures relative to the active or specified scope
Syntax	<pre>save_upf upf_file_name [-scope instance_name [-version string]</pre>	ie]
Arguments	upf_file_name	The UPF file to write.
	-scope instance_name	The scope relative to which the UPF commands are written.
Deprecated arguments	-version string	The UPF version of <i>upf_file_name</i> . See also <u>6.54</u> . This is a deprecated option; see also <u>6.1</u> and <u>Annex D</u> .
Return value	Return an empty string if	successful or raise a TCL_ERROR if not.

The **save_upf** command creates a UPF file that contains the power intent specified for a given scope. The power intent for that scope is written to file *upf_file_name*. The output file is generated after the power intent model has been constructed (see <u>8.3.2</u>.)

If **-scope** *instance_name* is specified, the power intent is written for the specified scope. It is an error if this scope does not exist. Otherwise, the power intent is written for the current scope.

The following also apply:

- a) Each invocation of **save_upf** generates a separate UPF output file.
- b) If **save_upf** is invoked for two scopes and one is an ancestor of the other, then the file generated for the ancestor shall contain a duplicate of the information in the file generated for the other.
- c) The following are equivalent:

```
save_upf <filename> -scope <instance>
and
set temp [set_scope <instance>]
save_upf <filename>
set scope $temp
```

Syntax example:

save_upf test_suite1_Jan14
-scope top/proc_1

6.37 set_design_attributes

Purpose	Apply attributes to model	s or instances	
Syntax	set_design_attributes [-models model_list] [-elements element_list] [-exclude_elements exclude_list] [-attribute {name value}]* [-is_leaf_cell [<true false="" ="">]] [-is_macro_cell [<true false="" ="">]]</true></true>		
	-models model_list	A list of models to be attributed.	
	-elements element_list	A list of rooted names: instances, named processes, sequential $regs$, or signal names.	
	-exclude_elements element_list	A list of rooted names: instances, named processes, sequential regs, or signal names to exclude from the <i>effective_element_list</i> (see 5.10).	
Arguments	<pre>-attribute {name value}</pre>	For the specified models or elements, associate the attribute <i>name</i> with the value of <i>value</i> . See <u>Table 4</u> .	
	-is_leaf_cell [<true <br="">FALSE>]</true>	If -is_leaf_cell is not specified at all, the default is FALSE . If -is_leaf_cell is specified without a value, the default value is TRUE . Equivalent to -attribute {UPF_is_leaf_cell <i>value</i> } (see <u>5.6</u>).	
	-is_macro_cell [<true false="" ="">]</true>	If -is_macro_cell is not specified at all, the default is FALSE . If -is_macro_cell is specified without a value, the default value is TRUE . Equivalent to -attribute {UPF_is_macro_cell value} (see <u>5.6</u>).	
Return value	Return an empty string if	successful or raise a TCL_ERROR if not.	

This command sets the specified attributes for models or elements. It is an error if **set_design_attributes** is specified

- a) with neither -models nor -elements; or
- b) with both -models and -elements; or
- c) with -exclude elements, but not -elements; or
- d) without at least one of -attribute, -is_leaf_cell, or -is_macro_cell.

A UPF_is_leaf_cell attribute value of "TRUE" on a model or instance prevents the -transitive processing for the descendants of the attributed model or instance for the following commands:

- connect_supply_set (see <u>6.12</u>)
- set_port_attributes (see <u>6.46</u>)
- set_retention (see 6.49)
- set_retention_elements (see <u>6.51</u>)
- find_objects (see 6.26)

A UPF_is_macro_cell attribute value of "TRUE" on a model or instance causes any ports of an instance of the model to be recognized as part of the lower boundary of the power domain containing that instance if the driver or receiver supply of that port is specified as an attribute and is neither identical to nor equivalent to the primary supply of the containing power domain (see <u>6.17</u>).

Examples

```
set_design_attributes -elements {lock_cache[0]}
    -attribute {UPF_is_leaf_cell TRUE}
set_design_attributes -models FIFO
    -attribute {UPF_is_leaf_cell TRUE}
set_design_attributes -models FIFO -is_leaf_cell
```

6.38 set_design_top

Purpose	Specify the design top module
Syntax	<pre>set_design_top design_top_module</pre>
Arguments	<i>design_top_module</i> The top module for which a UPF file was written.
Return value	Return an empty string.

The set_design_top command specifies the module for which this UPF file was written. See 4.2.7.

It is not an error if the instance to which this UPF file is applied is not an instance of the specified module. In particular, as long as the actual module has the same structure as the specified module, it may be possible to apply this UPF file to that module without errors. In this case, a tool may choose to issue a warning message.

Syntax example:

set_design_top ALU07

6.39 set_domain_supply_net [legacy]

Purpose	Set the default power and ground supply nets for a power domain		
Syntax	<pre>set_domain_supply_net domain_name -primary_power_net supply_net_name -primary_ground_net supply_net_name</pre>		
Arguments	domain_name	The domain where the default supply nets are to applied.	
	-primary_power_net supply_net_name	The primary power supply net.	
	-primary_ground_net supply_net_name	The primary ground net.	
Return value	Return a 1 if successful o	r raise a TCL_ERROR if not.	

This is a legacy command; see also 6.1 and Annex D.

The **set_domain_supply_net** command associates the power and ground supply nets with the primary supply set for the domain.

The primary supply set's power and ground functions for the specified domain are associated with the corresponding power and ground supply net.

It shall be an error if

- *domain_name* does not indicate a previously created power domain.
- The primary supply set for *domain_name* already has a primary power or ground function association.

This command is semantically equivalent to

```
proc set_domain_supply_net {dn pp sn1 pg sn2} {
    if { string equal $pp "-primary_power_net" \
        && string equal $pg "-primary_ground_net"}{
        create_supply_set set_name -function {power $sn1}
            -function {ground $sn2}
        associate_supply_set set_name -handle $dn.primary
        return 1
    } else {
        return -code TCL_ERROR \
            -errorinfo $einfo \
            $resulttext
    } }
```

where any *italicized* arguments are implementation defined.

Syntax example:

```
set_domain_supply_net PD1
-primary_power_net PG1
-primary_ground_net PG0
```

6.40 set_equivalent

Purpose	Declare that supply nets	or supply sets are electrically or functionally equivalent	
Syntax	set_equivalent [-function_only] [-nets supply_net_name_list] [-sets supply_set_name_list]		
Arguments	-function_only	Specifies that the supplies are functionally equivalent rather than electrically equivalent.	
	-nets supply_net_name_list	A list of supply port and/or supply net names that are equivalent.	
	-sets supply_set_name_list	A list of supply set names that are equivalent.	
Return value	Return an empty string if	successful or raise a TCL_ERROR if not.	

The set_equivalent command declares that two or more supplies are equivalent (see 4.4.3).

If **-function_only** is specified, then the supplies are declared to be *functionally equivalent* only; otherwise the supplies are declared to be *electrically equivalent*, which implies that they are also functionally equivalent.

If **-nets** is specified, the command defines equivalence for a list of supply ports and/or supply nets. If **-sets** is specified, the command defines equivalence for a list of supply sets and/or supply set handles. One or the other of these options, but not both, shall be specified.

Equivalence of supply ports and nets can affect the number of sources for a given supply network and whether resolution is required (see 9.1). Equivalence of supply sets and supply set handles can affect various commands whose semantics are based on supply set identity or equivalence, including **create_composite_domain** (see 6.13), **create_power_domain** (see 6.17), **set_isolation** (see 6.41), **set_level_shifter** (see 6.43), **set_repeater** (see 6.48), and **set_port_attributes** (see 6.46).

For the input to an implementation tool, it shall be an error if electrical equivalence has been specified for two nets/sets but the actual connections that cause the electrical equivalence are not present in the UPF or in the HDL. For the input to a simulation tool, the actual connections implementing electrical equivalence need not be specified.

Syntax example:

set_equivalent -nets { vss1 vss2 ground }
set_equivalent -function_only -nets { vdd_wall vdd_battery }
set equivalent -function only -sets { /sys/aon ss /mem/PD1.core ssh }

6.41 set_isolation

Purpose	Specify an isolation strat	egy	
Syntax	<pre>set_isolation strategy_name -domain domain_name [-elements element_list] [-exclude_elements exclude_list] [-source <source_domain_name source_supply_ref="" ="">] [-sink <sink_domain_name sink_supply_ref="" ="">] [-sink <sink_domain_name sink_supply_ref="" ="">] [-diff_supply_only [<true false="" ="">]] [-auplies_to_clamp <0 1 any Z latch value>] [-applies_to_clamp <0 1 any Z latch value>] [-applies_to_source_off_clamp <0 1 any Z latch value>] [-no_isolation] [-force_isolation] [-force_isolation] [-location <self automatic="" fanin="" faninout="" fanout="" other="" parent="" sibling="" ="">] [-clamp_value {< 0 1 any Z latch value>*}] [-isolation_signal signal_list [-isolation_sense {<high low="" ="">*}]]] [-isolation_supply_set supply_set_list] [-name_prefix string] [-name_suffix string] [-instance {{instance_name port_name}}*]] [-source_off_clamp {< 0 1 any Z latch value> [simstate_list]}] [-source_off_clamp {< 0 1 any Z latch value> [simstate_list]}] [-source_off_clamp {< 0 1 any Z latch value> [simstate_list]}] [-isolation_supply_set_supply_set_list] [-name_prefix string] [-name_prefix string] [-instance {{instance_name port_name}*] [-source_off_clamp {< 0 1 any Z latch value> [simstate_list]}] [-solation_power_net net_name] [-isolation_ground_net net_name] [-transitive [<true false="" ="">]] </true></high></self></true></sink_domain_name></sink_domain_name></source_domain_name></pre>		
	strategy_name	The name of the isolation strategy.	
	-domain aomain_name	A list of instances or ports to which the strategy potentially applies	R
	-exclude_elements exclude_list	A list of instances or ports to which the strategy does not apply.	R
	-source <source_domain_name source_supply_ref ></source_domain_name 	The rooted name of a supply set or power domain. When a domain name is used, it represents the primary supply of that domain.	R
Arguments	-sink <sink_domain_name <br="">sink_supply_ref></sink_domain_name>	The rooted name of a supply set or power domain. When a domain name is used, it represents the primary supply of that domain.	R
	-diff_supply_only [<true false="" ="">]</true>	Indicates whether ports connected to other ports with the same supply should be isolated. The default is -diff_supply_only FALSE if the option is not specified at all; if -diff_supply_only is specified without a value, the default value is TRUE .	R
	-use_equivalence [<true false="" ="">]</true>	Indicates whether to consider supply set equivalence. If -use_equivalence is not specified at all, the default is -use_equivalence TRUE ; if -use_equivalence is specified without a value, the default value is TRUE .	R
	-applies_to <inputs <br="">outputs both></inputs>	A filter that restricts the strategy to apply only to ports of a given direction.	R

	-applies_to_clamp <0 1 any Z latch value>	A filter that restricts the strategy to apply only to ports with a particular clamp value requirement.	R
	-applies_to_sink_off_ clamp <0 1 any Z latch value>	A filter that restricts the strategy to apply only to ports with a particular sink off clamp value requirement.	R
	-applies_to_source_off _clamp <0 1 any Z latch value>	A filter that restricts the strategy to apply only to ports with a particular source off clamp value requirement.	R
	-no_isolation	Specifies that isolation cells shall not be inserted on the specified ports.	R
	-force_isolation	Disables any implementation optimization involving isolation cells for a given strategy; used to force redundant isolation or to keep floating/ constant ports that have an isolation strategy defined for them.	R
Arguments	-location <self other="" ="" <br="">parent automatic fanout fanin faninout sibling></self>	The location in which inferred isolation cells are placed in the logic hierarchy, which determines the power domain in which they will exist. The default is self.	R
	-clamp_value {<0 1 any Z latch value>*}	The value(s) that the isolation cell can drive. The default is 0 .	R
	-isolation_signal signal_list	The control signal(s) that cause(s) the isolation cell to drive with the corresponding value(s) specified in -clamp_value .	R
	-isolation_sense { <high low>*}</high 	The active level(s) of the corresponding isolation signal(s) specified in -isolation_signal . The default is high .	R
	-isolation_supply_set supply_set_list	The supply set(s) that power the isolation cell for the corresponding control signal(s) specified in –isolation_signal .	R
	-name_prefix string -name_suffix string	The name format (prefix and suffix) for generated isolation instances or nets related to implementation of the isolation strategy.	R
	<pre>-instance {{instance_ name port_name}*}</pre>	The name of a technology leaf-cell instance and the name of the logic port that it isolates.	R
	-update	Indicates that this command provides additional information for a pre- vious command with the same <i>strategy_name</i> and <i>domain_name</i> and executed in the same scope.	R
Legacy arguments	-isolation_power_net net_name	This option specifies the supply net used as the power for the isolation logic inferred by this strategy. This is a legacy option; see also 6.1 and <u>Annex D</u> .	R
	-isolation_ground_net net_name	This option specifies the supply net used as the ground for the isolation logic inferred by this strategy. This is a legacy option; see also 6.1 and <u>Annex D</u> .	R

	-location fanin faninout sibling	The isolation cell is placed at all fanin locations (sources) of the port being isolated. The isolation cell is placed at all fanout locations (sinks) for each out- put port being isolated, or at all fanin locations (sources) for each input port being isolated. A new sibling is created into which the isolation cells are placed in the logic hierarchy. These are all deprecated options; see also <u>6.1</u> and <u>Annex D</u> .	R
Deprecated	-clamp_value { <any>}</any>	The -clamp_value option any . This is a deprecated option; see also <u>6.1</u> and <u>Annex D</u> .	R
arguments	-sink_off_clamp {<0 1 any Z latch value> [simstate_list]}	The -sink_off_clamp option specifies the clamp requirement when the sink domain is off. This is a deprecated option; see also 6.1 and <u>Annex D</u> .	R
	<pre>-source_off_clamp {<0 1 any Z latch value> [simstate_list]}</pre>	The -source_off_clamp option specifies the clamp requirement when the source domain is off. This is a deprecated option; see also 6.1 and <u>Annex D</u> .	R
	-transitive [<true false="" ="">]</true>	When -transitive is TRUE (the default), the command applies to the descendants of the elements. This is a deprecated option; see also 6.1 and <u>Annex D</u> .	
Return value	Return an empty string if	successful or raise a TCL_ERROR if not.	

The set_isolation command defines an isolation strategy for ports on the interface of a power domain (see 6.17). An isolation strategy is applied at the domain boundary, as required to ensure correct electrical and logical functionality when domains are in different power states.

-domain specifies the domain for which this strategy is defined.

-elements explicitly identifies a set of candidate ports to which this strategy potentially applies. The *element_list* may contain rooted names of instances or ports in the specified domain. If an instance name is specified in the *element_list*, it is equivalent to specifying all the ports of the instance in the *element_list*, but with lower precedence (see <u>5.8</u>). Any *element_lists* specified on the base command or any updates (see **-update**) of the base command are combined. If **-elements** is not specified in the base command or any update, every port on the interface of the domain is included in the *aggregate element list* (see <u>5.10</u>).

-exclude_elements explicitly identifies a set of ports to which this strategy does not apply. The *exclude_list* may contain rooted names of instances or ports in the specified domain. If an instance name is specified in the *exclude_list*, it is equivalent to specifying all the ports of the instance in the *exclude_list*. Any *exclude_list* specified on the base command or any updates of the base command are combined into the *aggregate_exclude_list* (see 5.10).

The arguments -source, -sink, -diff_supply_only, -applies_to, -applies_to_clamp, -applies_to_sink_off _clamp, and -applies_to_source_off_clamp serve as filters that further restrict the set of ports to which a given set_isolation command applies. The command only applies to those ports that satisfy all of the specified filters.

-source is satisfied by any port that is driven by logic powered by a supply set that matches (see -use_equivalence) the specified supply set, ignoring any isolation or level-shifting cells that have already been inferred or instantiated from an isolation or level-shifting strategy.

-sink is satisfied by any port that is received by logic powered by a supply set that matches (see -use_equivalence) the specified supply set, ignoring any isolation or level-shifting cells that have already been inferred or instantiated from an isolation or level-shifting strategy.

NOTE 1—A port that does not have a driver will never satisfy the **-source** filter. A port that does not have a receiver will never satisfy the **-sink** filter.

-diff_supply_only TRUE is satisfied by any port for which the driving logic and receiving logic are powered by supply sets that do not match (see -use_equivalence), or for which either driving or receiving or both supply sets cannot be determined. -diff_supply_only FALSE is satisfied by any port.

-use_equivalence specifies whether supply set equivalence is to be considered in determining when two supply sets match. If -use_equivalence is specified with the value *False*, the -source and -sink filters shall match only the named supply set; the -diff_supply_only TRUE filter shall be satisfied only if the driver supply and receiver supply of the port are not identical. Otherwise, the -source and -sink filters shall match the named supply set or any supply set that is equivalent to the named supply set; the -diff_supply_only TRUE filter shall be satisfied only if the driver supply and receiver supply of the port are neither identical nor equivalent.

-applies_to is satisfied by any port that has the specified mode. For upper boundary ports, this filter is satisfied when the direction of the port matches. For lower boundary ports, this filter is satisfied when the inverse of the direction of the port matches. For example, a lower boundary port with a direction OUT would satisfy the -applies_to IN filter, because an output from a lower boundary port is an input to this domain. -applies_to is always relative to the specified domain.

-applies_to_clamp, -applies_to_sink_off_clamp, and -applies_to_source_off_clamp are satisfied by any port that has the specified value for the UPF_clamp_value, UPF_sink_off_clamp, or UPF_source_off_clamp port attribute, respectively.

The *effective_element_list* (see <u>5.10</u>) for this command consists of all the port names in the *aggregate_element_list* that are not also in the *aggregate_exclude_list* and that satisfy all of the filters specified in the command. If a port in the *effective_element_list* is not on the interface of the specified domain, it shall not be isolated.

If a given port name is referenced in the *effective_element_list* of more than one isolation strategy of a given domain, the precedence rules (see <u>5.8</u>) determine which of those strategies actually apply to that port name. If the precedence rules identify multiple strategies that apply to the same port name, then those strategies shall each have a **-sink** filter that matches the receiving supply of a different sink domain for the specified port. It shall be an error if the precedence rules identify multiple strategies that apply to the same port name such that more than one strategy applies to the same sink domain for that port.

If **-no_isolation** is specified, then isolation is not inferred for any port in the *effective_element_list*.

If **-force_isolation** is specified, then isolation is inferred for each port in the *effective_element_list* and the inferred isolation cells are not to be optimized away, even if such optimization does not change the behavior of the design.

If neither **-no_isolation** nor **-force_isolation** is specified, then isolation is inferred for each port in the *effective_element_list*, and implementation tools are free to optimize away isolation cells that are redundant provided that such optimization does not change the behavior of the design.

-location defines where the isolation cells are placed in the logic hierarchy and therefore the power domain into which they are inserted, as follows:

self—the isolation cell is placed inside the domain whose interface port is being isolated (the default).

other—the isolation cell is placed in the parent for ports on the interface of the domain that connect to the parent, and in the child for ports on the interface of the domain that connect to a child.

parent—the isolation cell is placed in the parent of the instance whose interface port is being isolated.

fanout—the isolation cell is placed at all fanout locations (receiving logic) of the port being isolated.

automatic—the implementation tool is free to choose any of the locations self, parent, or other.

If **-location fanout** is specified, the isolation cell shall be inserted at the port on the domain boundary that is closest to the receiving logic. If the receiving logic is in a macro cell instance, the isolation cell shall be inserted in the domain that contains the macro cell instance; otherwise the isolation cell shall be inserted in the domain that contains the receiving logic.

If **-location automatic** is specified, and a second isolation strategy is also applied to this port by the other power domain sharing this interface, the location chosen by the tool shall be such that the isolation cell contributed by the source domain is placed closer to the driving logic and the isolation cell contributed by the sink domain is placed closer to the receiving logic.

If any pair of isolation cells are inferred from two different isolation strategies for ports of two different power domains along the same path from a driver to a receiver, and the **–location** specified results in both cells being inserted into the same domain, then the two isolation cells shall be inserted such that the isolation cell contributed by the source domain is placed closer to the driving logic and the isolation cell contributed by the sink domain is placed closer to the receiving logic.

If two or more isolation strategies apply to the same port on the interface of a power domain, such that multiple isolation cells need to be inferred for different paths from that port to a receiving domain, the **-location** specified explicitly or implicitly for each of those strategies shall be such that the various isolation cells can be inserted without splitting the port into multiple ports. It shall be an error if multiple isolation strategies for the same port cannot be implemented without duplicating the port.

The -clamp_value, -isolation_signal and -isolation_sense, and -isolation_supply_set options are each specified as a single value or a list. If any of these options specify a list, then all lists specified for these options shall be of the same length and any single value specified is treated as a list of values of the same length. The tuples formed by associating the positional entries from each list shall be used to define separate isolation requirements for the strategy. These tuples are applied to the isolation cell from the isolation cell's data input port to its data output port in the order in which they appear in each list. The output of the isolation cell shall be the right-most value in the -clamp_value list whose corresponding isolation signal is active.

-clamp_value specifies the value of the inferred isolation cell's output when isolation is enabled. The specification may be a single value or a list of values. Any of the following may be specified:

- **(**the logic value 0)
- 1 (the logic value 1)
- **Z** (the logic value Z)

latch (the value of the non-isolated port when the isolation signal becomes active)

value specifies a value that is legal for the type of the port, e.g., 255 might be specified for an integer-typed port (perhaps constrained to an unsigned 8-bit range).

If -clamp_value is not specified, it defaults to 0.

Verification shall issue an error when a UPF_sink_off_clamp, UPF_source_off_clamp, or UPF clamp value requirement is violated.

-isolation_signal identifies the control signal for each clamp value specified by -clamp_value.

-isolation_sense specifies the value that enables isolation, for each signal specified by -isolation_signal.

-isolation_supply_set specifies the supply set(s) that shall be used to power the inferred isolation cell. The isolation supply set(s) specified by -isolation_supply_set are implicitly connected to the isolation logic inferred by this command. If -isolation_supply_set is not specified, the default_isolation supply of the power domain in which the inferred cell will be located is used as the isolation supply. For example, if set_isolation is specified with -location parent and -isolation_supply_set is not specified, then the default_isolation supply set of the parent domain is used.

-name_prefix specifies the substring to place at the beginning of any generated name implementing this strategy.

-name_suffix specifies the substring to place at the end of any generated name implementing this strategy.

-instance specifies that the isolation functionality exists in the HDL design and *instance_name* denotes the instance providing part or all of this functionality. An *instance_name* is a simple name or hierarchical name rooted in the current scope. If an empty string appears as an *instance_name*, this indicates that an instance was created and then optimized away. Such an instance should not be reinferred or reimplemented by subsequent tool runs.

In this case, the following also apply:

- Isolation enable signal(s) are automatically connected to one or more ports of an instance of a cell defined by the library command define_isolation_cell (see 7.4). If the strategy specifies multiple isolation enable signals, then the cell shall also be defined with both the -enable option and the -aux_enables option (see 7.4), the first isolation enable signal shall be connected to the port specified by the -enable option, and the rest of the signals shall be connected to the ports specified by the -aux enables option in the same order.
- If the strategy specifies a single isolation supply set, the supply nets of the set shall be automatically connected to the primary supply ports of the isolation cell. If the strategy specifies multiple isolation supply sets, the isolation enable ports shall have related power, ground, and bias port attributes (see 6.45 and 6.46), and the supply nets of the isolation supply set corresponding to each isolation enable signal shall be automatically connected to the supply ports matching the related power, ground, and bias ports of the isolation enable port (see 7.4).
- If there are no supply ports on the instance, then the isolation supply set(s) specified in the strategy shall be implicitly connected to the instance.
- It is an error if there is a single isolation enable signal and there is more than one port on the library cell of the instance defined as isolation enable pin or aux enable pin (see 7.4).

-update adds information to the base command executed in the same scope. When specified with -update, -elements and -exclude_elements are additive: the set of instances or ports in the *aggregate_elements_list* is the union of all -elements specifications given in the base command and any update of this command, and the *aggregate_exclude_list* is the union of all -exclude_elements specifications given in the base command and any update of this command, and any update of this command.

Tools shall not use information about system power states to avoid inserting isolation as directed by these strategies. However, tools may optionally use information about system power states to issue a warning that certain strategies appear to be unnecessary.

The following also apply:

- This command never applies to inout ports.
- It is erroneous if an isolation strategy isolates its own control signal.
- It shall be an error if -no_isolation is specified along with any of the following: -force_isolation,
 -isolation_signal, -isolation_sense, -instance, -location, -name_prefix, -name_suffix,
 -isolation_supply_set, -isolation_power_net, or -isolation_ground_net.
- It shall be an error if the isolation supply set is not defined for a strategy and the domain in which the inferred isolation cell is located does not have a **default isolation** supply set.

NOTE 2—To specify an isolation strategy for a port P on the lower boundary of a power domain D (see 4.3.1), a **set_isolation** command can specify -domain D and specify the port name I/P, where I is the hierarchical name of an instance that is instantiated in domain D but is not in the extent of domain D, and P is the simple name of the port of that instance. The combination of the **-domain** specification and the hierarchical port name makes it clear this reference is to the HighConn of the specified port, which is part of the lower boundary of the domain D.

NOTE 3—The *exclude_list* in **-exclude_elements** can specify instances or ports that have not already been explicitly or implicitly specified via an explicit or implied *element_list*.

NOTE 4—If a **-diff_supply_only**, **-source**, or **-sink** argument is used and instances are included in designs with different power distribution or connectivity, the evaluation of the need for isolation may vary and cause a change in the logical function of a block.

NOTE 5—Isolation clamp value port properties can be annotated in HDL using the attributes shown in 5.6. The same attributes may be specified using the **set_port_attributes** command in 6.46.

NOTE 6-It is not an error if multiple isolation strategies apply to a connection from one domain to another domain.

Syntax example:

```
set_isolation parent_strategy
  -domain pda
  -elements {a b c d}
  -isolation_supply_set {pda_isolation_supply}
    -clamp_value {1}
  -applies_to outputs -sink pdb
set_isolation parent_strategy -update
    -domain pda
    -isolation_signal cpu_iso
```

-isolation_sense low -location parent

6.42 set_isolation_control [deprecated]

This is a deprecated command; see also 6.1 and Annex D.

6.43 set_level_shifter

Purpose	Specify a level-shifter strategy		
Syntax	<pre>set_level_shifter strategy_name -domain domain_name [-elements element_list] [-exclude_elements exclude_list] [-source <source_domain_name source_supply_ref="" ="">] [-sink <sink_domain_name sink_supply_ref="" ="">] [-sink <sink_domain_name sink_supply_ref="" ="">] [-use_equivalence [<true false="" ="">]] [-use_equivalence [<true false="" ="">]] [-applies_to <inputs both="" outputs="" ="">] [-threshold <value list="" ="">] [-threshold <value list="" ="">] [-location <self automatic="" fanin="" faninout="" fanout="" other="" parent="" sibling="" ="">] [-input_supply_set_supply_set_ref] [-internal_supply_set_supply_set_ref] [-internal_supply_set_supply_set_ref] [-instance {{instance_name port_name}*}] [-update] [-transitive [<true false="" ="">]] </true></self></value></value></inputs></true></true></sink_domain_name></sink_domain_name></source_domain_name></pre>		
	strategy_name	The name of the level-shifter strategy.	
	-domain domain_name	The domain for which this strategy is defined.	
Arguments	-elements element_list	A list of instances or ports to which the strategy potentially applies.	R
	-exclude_elements exclude_list	A list of instances or ports to which the strategy does not apply.	R
	-source <source_domain_name source_supply_ref></source_domain_name 	The rooted name of a supply set or power domain. When a domain name is used, it represents the primary supply of that domain.	R
	-sink <sink_domain_name <br="">sink_supply_ref></sink_domain_name>	The rooted name of a supply set or power domain. When a domain name is used, it represents the primary supply of that domain.	R
	-use_equivalence [<true false="" ="">]</true>	Indicates whether to consider supply set equivalence. If -use_equivalence is not specified at all, the default is -use_equivalence TRUE ; if -use_equivalence is specified without a value, the default value is TRUE .	R
	-applies_to <inputs <br="">outputs both></inputs>	A filter that restricts the strategy to apply only to ports of a given direction.	R
	-rule <low_to_high <br="">high_to_low both></low_to_high>	A filter that restricts the strategy to apply only to ports that require a given level-shifting direction. The default is both .	R
	-threshold <value <br="">list></value>	A filter that restricts the strategy to apply only to ports that involve a voltage difference above a certain threshold. The default is 0.	R
	-no_shift	Specifies that level-shifter cells shall not be inserted on the specified ports.	R
	-force_shift	Disables any implementation optimization involving level-shifter cells for a given strategy.	R

Arguments	-location <self other="" ="" <br="">parent automatic fanout fanin faninout sibling></self>	The location in which inferred level-shifter cells are placed in the logic hierarchy, which determines the power domain in which they will exist. The default is self .	R
	-input_supply_set supply_set_ref	The supply set used to power the input portion of the level-shifter.	R
	-output_supply_set supply_set_ref	The supply set used to power the output portion of the level-shifter.	R
	-internal_supply_set supply_set_ref	The supply set used to power internal circuits within the level-shifter.	R
	-name_prefix string -name_suffix string	The name format (prefix and suffix) for generated level-shifter instances or nets related to implementation of the level-shifting strategy.	R
	<pre>-instance {{instance_name port_name}*}</pre>	The name of a technology library leaf-cell instance and the name of the logic port that it level-shifts.	R
	-update	Indicates that this command provides additional information for a previous command with the same <i>strategy_name</i> and <i>domain_name</i> and executed in the same scope.	R
	-threshold < <i>list</i> >	<i>list</i> is a matrix of values. This is a deprecated option; see also 6.1 and <u>Annex D</u> .	R
Deprecated arguments	-location fanin faninout sibling	The level-shifter cell is placed at all fanin locations (sources) of the port being shifted. The level-shifter cell is placed at all fanout locations (sinks) for each output port being shifted, or at all fanin locations (sources) for each input port being shifted. A new sibling is created into which the level-shifter cells are placed in the logic hierarchy. These are all deprecated options; see also <u>6.1</u> and <u>Annex D</u> .	R
	-transitive [<true false="" ="">]</true>	When -transitive is TRUE (the default), the command applies to the descendants of the elements. This is a deprecated option; see also <u>6.1</u> and <u>Annex D</u> .	
Return value	Return an empty string it	f successful or raise a TCL_ERROR if not.	

The set_level_shifter command defines a level-shifting strategy for ports on the interface of a power domain (see 6.17). A level-shifter strategy is applied at the domain boundary, as required to correct for voltage differences between driving and receiving supplies of a port.

-domain specifies the domain for which this strategy is defined.

-elements explicitly identifies a set of candidate ports to which this strategy potentially applies. The *element_list* may contain rooted names of instances or ports in the specified domain. If an instance name is specified in the *element_list*, it is equivalent to specifying all the ports of the instance in the *element_list* but with lower precedence (see <u>5.8</u>). Any *element_lists* specified on the base command or any updates (see **-update**) of the base command are combined. If **-elements** is not specified in the base command or any update, every port on the interface of the domain is included in the *aggregate_element_list* (see <u>5.10</u>).

-exclude_elements explicitly identifies a set of ports to which this strategy does not apply. The *exclude_list* may contain rooted names of instances or ports in the specified domain. If an instance name is specified in the *exclude_list*, it is equivalent to specifying all the ports of the instance in the *exclude_list*. Any *exclude_list* specified on the base command or any updates of the base command are combined into the *aggregate exclude list* (see <u>5.10</u>).

The arguments **-source**, **-sink**, **-applies_to**, **-rule**, and **-threshold** serve as filters that further restrict the set of ports to which a given **set_level_shifter** command applies. The command only applies to those ports that satisfy all of the specified filters.

-source is satisfied by any port that is driven by logic powered by a supply set that matches (see **-use_equivalence**) the specified supply set, ignoring any isolation or level-shifting cells that have already been inferred or instantiated from an isolation or level-shifting strategy.

-sink is satisfied by any port that is received by logic powered by a supply set that matches (see -use_equivalence) the specified supply set, ignoring any isolation or level-shifting cells that have already been inferred or instantiated from an isolation or level-shifting strategy.

NOTE 1—A port that does not have a driver will never satisfy the **-source** filter. A port that does not have a receiver will never satisfy the **-sink** filter.

-use_equivalence specifies whether supply set equivalence is to be considered in determining when two supply sets match. If -use_equivalence is specified with the value *False*, the -source and -sink filters shall match only the named supply set. Otherwise, the -source and -sink filters shall match the named supply set or any supply set that is equivalent to the named supply set.

-applies_to is satisfied by any port that has the specified mode. For upper boundary ports, this filter is satisfied when the direction of the port matches. For lower boundary ports, this filter is satisfied when the inverse of the direction of the port matches. For example, a lower boundary port with a direction OUT would satisfy the -applies_to IN filter, because an output from a lower boundary port is an input to this domain. -applies_to is always relative to the specified domain.

-rule is satisfied by any port for which the driving and receiving logic have the specified voltage relationship. If **low_to_high** is specified, a given port satisfies this filter if the voltage of its driver supply is less than the voltage of its receiver supply. If **high_to_low** is specified, a given port satisfies this filter if the voltage of its driver supply is greater than the voltage of its receiver supply. If **high_to_low** is specified, a given port satisfies this filter if the specified, a given port satisfies this filter if the voltage of its driver supply is greater than the voltage of its receiver supply. If **-rule both** is specified, a given port satisfies this filter if would satisfy either **-rule low_to_high** or **-rule high_to_low**.

-threshold is satisfied by any port for which the magnitude of the difference between the driver and receiver supply voltages can exceed a specified threshold value. The nominal power and ground of the port's driver supply are compared with the nominal power and ground of the port's receiver supply. This option requires tools to use information defined in power states of the supplies involved in a given interconnection between objects with different supplies. If **-threshold** is not specified, it defaults to 0, which ensures that a level-shifter will be inserted for a given port if there is any voltage difference.

The -threshold value is evaluated as shown in the following pseudo code:

next B next A return (NOT REQUIRED)

The *effective_element_list* (see 5.10) for this command consists of all the port names in the *aggregate_element_list* that are not also in the *aggregate_exclude_list* and that satisfy all of the filters specified in the command. If a port in the *effective_element_list* is not on the interface of the specified domain, it shall not be level-shifted.

If a given port name is referenced in the *effective_element_list* of more than one level-shifting strategy of a given domain, the precedence rules (see <u>5.8</u>) determine which of those strategies actually apply to that port name. If the precedence rules identify multiple strategies that apply to the same port name, then those strategies shall each have a **-sink** filter that matches the receiving supply of a different sink domain for the specified port. It shall be an error if the precedence rules identify multiple strategies that apply to the same port name such that more than one strategy applies to the same sink domain for that port.

If **-no_shift** is specified, then level-shifting is not inferred for any port in the *effective_element_list*.

If **-force_shift** is specified, then level-shifting is inferred for each port in the *effective_element_list* and the inferred level-shifting cells are not to be optimized away, even if such optimization does not change the behavior of the design.

If neither **-no_shift** nor **-force_shift** is specified, then level-shifting is inferred for each port in the *effective_element_list*, and implementation tools are free to optimize away level-shifting cells that are redundant provided that such optimization does not change the behavior of the design.

-location defines where the level-shifter cells are placed in the logic hierarchy and therefore the power domain into which they are inserted, as follows:

self—the level-shifter cell is placed inside the domain whose interface port is being shifted (the default).

other—the level-shifter cell is placed in the parent for ports on the interface of the domain that connect to the parent, and in the child for ports on the interface of the domain that connect to a child.

parent—the level-shifter cell is placed in the parent of the element whose interface port is being shifted.

fanout—the level-shifter cell is placed at all fanout locations (receiving logic) of the port being shifted.

automatic—the implementation tool is free to choose any of the locations self, parent, or other.

If **-location fanout** is specified, the level-shifter cell shall be inserted at the port on the domain boundary that is closest to the receiving logic.

If the port at which the level-shifter is inserted is connected to the input or output of an isolation cell, or is connected to the output of one isolation cell and the input of another isolation cell, the level-shifter is inserted either immediately before, or immediately after, or between the isolation cell(s), as appropriate, to achieve the best match between any explicitly specified input/output supplies of the strategy and the actual driver/receiver supplies at each location.

If multiple level-shifter strategies are defined that would insert a level-shifter at the same domain boundary, any of those level-shifter strategies can be applied in any of the preceding locations, in either domain, either singly or in combination. If two potential solutions match the driving and receiving supplies equally well, the solution that applies a level-shifting strategy contributed by a domain closer to the receiving domain shall be used.

For a port on the boundary between two domains, if neither domain explicitly defines a level-shifter strategy that applies to that port, then a default level-shifter strategy is implicitly defined for the LowConn side of the port, on the upper boundary of the lower domain. The default level-shifter strategy is as follows:

```
set_level_shifter -domain <domain name> -elements <port name> -rule both
    -threshold 0
```

-input_supply_set specifies the supply set connected to input supply ports of the level-shifter (see <u>7.4</u>). The default is the supply of the logic driving the level-shifter input. The default is used if and only if that supply set is available in the domain in which the level-shifter will be located. It shall be an error if the default supply set is required but is not available.

-output_supply_set specifies the supply set connected to the output supply ports of the level-shifter (see <u>7.4</u>). The default is the supply of the logic receiving the level-shifter output. The default is used if and only if that supply set is available in the domain in which the level-shifter will be located. It shall be an error if the default supply set is required but is not available.

Default input and output supply set definitions apply only if exactly one level-shifter strategy applies to a given port, all drivers of that port have equivalent supplies, and all receivers of that port have equivalent supplies. For more complex cases, the required supply sets should be explicitly specified.

If the level-shifter strategy is mapped to a library cell that requires only a single supply, then explicit specification of an input supply set is not required, any explicit input supply set specification is ignored, and the default input supply set does not apply; only the output supply set is used.

-internal_supply_set specifies the supply set that shall be used to provide power to supply ports that are not related to the inputs or outputs of the level-shifter. There is no default supply set defined for -internal_supply_set.

-name_prefix specifies the substring to place at the beginning of any generated name implementing this strategy.

-name_suffix specifies the substring to place at the end of any generated name implementing this strategy.

-instance specifies that the level-shifter functionality exists in the HDL design, and *instance_name* denotes the instance providing part or all of this functionality. An *instance_name* is a simple name or hierarchical name rooted in the current scope. If an empty string appears as an *instance_name*, this indicates that an instance was created and then optimized away. Such an instance should not be reinferred or reimplemented by subsequent tool runs.

-update adds information to the base command executed in the same scope. When specified with -update, -elements and -exclude_elements are additive: the set of instances or ports in the *aggregate_elements_list* is the union of all -elements specifications given in the base command and any update of this command, and the *aggregate_exclude_list* is the union of all -exclude_elements specifications given in the base command and any update of this command.

The following also apply:

- This command never applies to inout ports.
- The simstate semantics of all implicitly connected supply sets apply to the output of a level-shifter.
- It shall be an error if -no_shift is specified along with any of the following: -force_shift, -instance, -location, -name_prefix, -name_suffix, -input_supply_set, -output_supply_set, or -internal_supply_set.

It shall be an error if there is a connection between a driver and receiver and all of the following apply:

- The supplies powering the driver and receiver are at different voltage levels.
- A level-shifter is not specified for the connection using a level-shifter strategy.
- A level-shifter cannot be inferred for the connection by analysis of the power states of the supplies to the driver and receiver.

NOTE 2—To specify a level-shifting strategy for a port P on the lower boundary of a power domain D, a **set_level_shifter** command can specify -domain D and specify the port name I/P, where I is the hierarchical name of an instance that is instantiated in domain D but is not in the extent of domain D, and P is the simple name of the port of that instance. The combination of the **-domain** specification and the hierarchical port name makes it clear this reference is to the HighConn of the specified port, which is part of the lower boundary of the domain D.

NOTE 3—The *exclude_list* in **-exclude_elements** can specify instances or ports that have not already been explicitly or implicitly specified via an explicit or implied *element_list*.

NOTE 4—It is not an error if multiple level-shifting strategies apply to a connection from one domain to another domain.

Syntax example:

```
set_level_shifter shift_up
   -domain PowerDomainZ
   -applies_to inputs -source PowerDomainX.ss1
   -threshold 0.02
   -rule both
set_level_shifter TurnOffDefaultLS -domain PD -no_shift
//this turns off inference of a default level-shifter for ports on the
//upper boundary of domain PD
```

6.44 set_partial_on_translation

Purpose	Define the translation of PARTIAL_ON	
Syntax	set_partial_on_translation <off full_on="" =""></off>	
Arguments	OFF FULL_ON The value to use in place of PARTIAL_ON .	
Return value	Return the setting of the translation if successful or raise a TCL_ERROR if not.	

This command defines the translation of **PARTIAL_ON** to **FULL_ON** or **OFF** for purposes of evaluating the power state of supply sets and power domains. The state of a supply *set* is evaluated after **PARTIAL_ON** is translated to **FULL_ON** or **OFF** for each supply *net* in the set.

It shall be an error if this command is invoked with different values in the same UPF description.

Syntax example:

set_partial_on_translation FULL_ON

6.45 set_pin_related_supply [deprecated]

This is a deprecated command; see also 6.1 and Annex D.

6.46 set_port_attributes

Purpose	Define information on ports	
Syntax	set_port_attributes [-model name] [-elements element_lit [-exclude_elements et [-ports port_list] [-exclude_ports port_ [-applies_to <inputs] [-attribute {name valit [-clamp_value <0 1 [-sink_off_clamp <0 [-source_off_clamp <0 [-source_off_clamp <0 [-source_off_clamp <0 [-source_off_clamp <0 [-source_off_clamp <0 [-ereciver_supply supp [-receiver_supply supp [-receiver_supply supp [-receiver_supply supp [-related_power_por [-related_power_pors [-feedthrough] [-unconnected] [{-domains domain_lit [{-exclude_domains of [-repeater_supply supp [-transitive [<true]< th=""><th>st] lement_exclude_list] exclude_list] outputs both>] ue}]* any Z latch value>] 1 any Z latch value>] 1 any Z latch value>] 1 any Z latch value>] 2 1 any Z latch value>] y_set_ref] pply_set_ref] lue] t supply_port_name] supply_port_name] supply_port_name_list] list [-applies_to <inputs both="" ="">]}] domain_list [-applies_to <inputs both="" ="">]}] pply_set_ref] FALSE>]]</inputs></inputs></th></true]<></inputs] 	st] lement_exclude_list] exclude_list] outputs both>] ue}]* any Z latch value>] 1 any Z latch value>] 1 any Z latch value>] 1 any Z latch value>] 2 1 any Z latch value>] y_set_ref] pply_set_ref] lue] t supply_port_name] supply_port_name] supply_port_name_list] list [-applies_to <inputs both="" ="">]}] domain_list [-applies_to <inputs both="" ="">]}] pply_set_ref] FALSE>]]</inputs></inputs>
	-model name	A module or library cell whose ports are to be attributed.
	-exclude_elements	A list of instances whose ports are to be attributed. A list of instances whose ports are to be excluded from the command.
	element_exclude_list	
	<pre>-ports port_list</pre>	A list of simple names of ports to be attributed.
	<pre>-exclude_ports port_exclude_list</pre>	A list of ports to be excluded from the command.
	-applies_to <inputs <br="">outputs both></inputs>	Indicates whether the specified input ports, output ports, or both are to be attributed.
	<pre>-attribute {name value}</pre>	The attribute <i>name</i> and <i>value</i> pair to be associated with the specified ports.
Arguments	-clamp_value <0 1 any Z latch value>	The clamp requirement. Equivalent to -attribute {UPF_clamp_value value} (see <u>5.6</u>).
	-sink_off_clamp <0 1 any Z latch value>	The clamp requirement when the sink domain is off. Equivalent to -attribute {UPF_sink_off_clamp_value value} (see <u>5.6</u>).
	-source_off_clamp <0 1 any Z latch value>	The clamp requirement when the source domain is off. Equivalent to -attribute {UPF_source_off_clamp_value value} (see <u>5.6</u>).
	-driver_supply supply_set_ref	The supply set used by drivers of the port. Equivalent to -attribute {UPF_driver_supply <i>supply_set_ref</i> } (see <u>5.6</u>).
	-receiver_supply supply_set_ref	The supply set used by receivers of the port. Equivalent to -attribute {UPF_receiver_supply <i>supply_set_ref</i> } (see <u>5.6</u>).
	<pre>-pg_type pg_type_value</pre>	The <i>pg_type</i> port. Equivalent to -attribute {UPF_pg_type <i>pg_type_value</i> } (see <u>5.6</u>).

Arguments	<pre>-related_power_port supply_port_name</pre>	The power port for the attributed port. Equivalent to -attribute { UPF_related_power_port <i>supply_port_name</i> } (see <u>5.6</u>).
	<pre>-related_ground_port supply_port_name</pre>	The ground port for the attributed port. Equivalent to -attribute { UPF_related_ground_port <i>supply_port_name</i> } (see <u>5.6</u>).
	<pre>-related_bias_ports supply_port_name_list</pre>	The bias port(s) for the attributed port. Equivalent to -attribute { UPF_related_bias_ports <i>supply_port_name_list</i> } (see <u>5.6</u>).
	-feedthrough	Indicates that the specified ports are connected together internally to form a feedthrough. Equivalent to -attribute {UPF_feedthrough TRUE} (see <u>5.6</u>).
	-unconnected	Indicates that the specified ports are not connected at all internally. Equivalent to -attribute {UPF_unconnected TRUE} (see <u>5.6</u>).
Deprecated arguments	<pre>{-domains domain_list [-applies_to <inputs both="" outputs="" ="">]}</inputs></pre>	A list of domains whose ports are to be attributed. This is a deprecated option; see also 6.1 and Annex D.
	{-exclude_domains domain_list [-applies_to <inputs <br="">outputs both>]}</inputs>	A list of domains whose ports are excluded from being attributed. This is a deprecated option; see also 6.1 and Annex D.
	-repeater_supply supply_set_ref	The supply set used by a repeater driving the port. This is a deprecated option; see also 6.1 and Annex D.
	-transitive [<true <br="">FALSE>]</true>	If -transitive is not specified at all, the default is -transitive TRUE . If -transitive is specified without a value, the default value is TRUE . This is a deprecated option; see also <u>6.1</u> and <u>Annex D</u> .
Return value	Return an empty string if	successful or raise a TCL_ERROR if not.

The **set_port_attributes** command specifies information associated with ports of models or instances. Certain predefined attributes identify a port's related supplies and in doing so may define the lower boundary of a power domain; other predefined attributes provide information relevant to isolation and level-shifting insertion.

User-defined attributes may also be associated with a port. The meaning of a user-defined attribute is not specified by this standard.

The set of ports attributed is determined as follows:

- a) A set of candidate ports is first identified. This set includes the following:
 - 1) If **-elements** is specified, all ports of each instance named in the elements list are included in the candidate set, including any logic ports inferred from **create_logic_port** (see <u>6.16</u>), but excluding any supply ports.
 - 2) If **-ports** is specified, each port named in the ports list is included in the candidate set.
- b) The candidate set is then restricted to those ports that satisfy any filters specified. A port is removed from the candidate set if:
 - 1) The port name appears in the **-exclude_ports** list.
 - 2) The port is a port on an instance named in the **-exclude_elements** list.
 - 3) The port direction is not consistent with the direction identified by the **-applies_to** option.
- c) The resulting restricted set is the set of ports to be attributed.

If **-model** is specified, the port attributes are applied to the selected ports of each instance of the model. In this case, only names that are declared in the model may be referenced in arguments to this command and all names are interpreted relative to the topmost scope of the model.

-model and -attribute can be used together to specify attributes for ports of a hard IP. For example, if ports of the hard IP are connected to each other by the same metal wire, i.e., a feedthrough connection, they should have the UPF_feedthrough attribute set to TRUE. If a port is not connected to any logic inside the hard IP, it should have the UPF_unconnected attribute set to TRUE. For more details, see <u>Annex G</u>.

-clamp_value defines the UPF_clamp_value attribute, which specifies the clamp value to be used if this port has an isolation strategy applied to it.

-sink_off_clamp defines the UPF_sink_off_clamp attribute, which specifies the clamp requirement when the supply set connected to the sink is in a power state with a corresponding simstate of CORRUPT.

-source_off_clamp defines the UPF_source_off_clamp attribute, which specifies the clamp requirement when the supply set connected to the source is in a power state with a corresponding simstate of CORRUPT.

When a user-defined clamp *value* is specified for **UPF_sink_off_clamp** or **UPF_source_off_clamp**, it shall be a legal value for the type of the port. A clamp value of **any** specifies any clamp value legal for the port type is allowed. If the port needs to be isolated in a given context, the specific clamp value to use shall be specified in a **set_isolation** command (see <u>6.41</u>).

-driver_supply and -receiver_supply define the attributes UPF_driver_supply or UPF_receiver_supply, respectively. These attributes can be used to specify the driver supply of a macro cell output port or the receiver supply of a macro cell input port. They can also be used to specify the driver supply of external logic driving a primary input or to specify the receiver supply of external logic receiving a primary output.

When the **UPF_driver_supply** attribute is defined for a port, it specifies the driver supply of the logic driving the port. If the driving logic is not within the logic design starting at the design root, it is presumed the specified driver supply is the supply for the driver logic; therefore, the port is corrupted when the driver supply is in a simstate other than **NORMAL**. For a port with the attribute **UPF_driver_supply**, when that port has a single source and the driving logic is present within the logic design starting at the design root, it shall be an error if the supply of the driving logic is not the same as, or equivalent to, the specified driver supply.

When the **UPF_receiver_supply** attribute is defined for a port, it specifies the receiver supply of the logic receiving the port. If the receiving logic is not within the logic design starting at the design root, it is presumed the specified receiver supply is the supply for the receiving logic. For a port with the attribute **UPF_receiver_supply**, when that port has a single sink and the receiving logic is present within the logic design, it shall be an error if the supply of the receiving logic is not the same as, or equivalent to, the specified receiver supply.

If **UPF_driver_supply** is not defined for a primary input port or **UPF_receiver_supply** is not defined for a primary output port, the default driver supply or receiver supply, respectively, is an anonymous supply set that is not equivalent to any other supply set.

-pg_type defines the UPF_pg_type attribute on a supply port for use with automatic connection semantics. *pg_type_value* is a string denoting the supply port type.

NOTE—**UPF_pg_type** only applies to supply ports and is the only predefined attribute that applies to supply ports. All other attributes apply to logic ports.

If any of **-related_power_port**, **-related_ground_port**, or **-related_bias_ports** is specified, an implicit supply set is created consisting of the supply nets connected to the specified ports. If **-related_power_port** *supply_port_name* and **-related_ground_port** *supply_port_name* are specified, the specified *supply_port_name* and **-related_ground_port** *supply_port_name* are specified, the specified *supply_port_name* shall be used as the power and ground functions, respectively, of the implicit supply set. If **-related_bias_ports** *supply_port_name_list* is specified, each port in the *supply_port_list* shall have a *pg_type* of nwell, pwell, deepnwell, or deeppwell, and each port shall be used as the appropriate bias function of the implicit supply set, as indicated by the value of the associated attribute.

If the port being attributed is in mode, the related ports specify the UPF_receiver_supply attribute of the port being attributed, as if the implicitly created supply set were specified as the -receiver_supply argument. If the port being attributed, as if the implicitly created supply set were specified as the -receiver_supply attribute of the port being attributed, as if the implicitly created supply set were specified as the -receiver_supply attribute of the port being attributed, as if the implicitly created supply set were specified as the -receiver_supply argument. If the port being attributed is inout mode, the related ports specify both the UPF_receiver_supply and the UPF_driver_supply attributes of the port being attributed, as if the implicitly created supply and the UPF_driver_supply attributes of the port being attributed, as if the implicitly created supply and the -driver_supply attributes of the port being attributed, as if the implicitly created supply and the -driver_supply attributes of the port being attributed, as if the implicitly created supply and the -driver_supply attributes of the port being attributed, as if the implicitly created supply set were specified as both the -receiver_supply and the -driver_supply arguments.

By the previous definition, related supplies always refer to the driver and receiver supplies of the logic inside a module.

-feedthrough defines the UPF_feedthrough attribute, which identifies a set of ports on the interface of a module or cell that are directly connected to each other inside the module or cell and therefore create a feedthrough through the module or cell.

-unconnected defines the **UPF_unconnected** attribute, which identifies a set of ports on the interface of a module or cell that are not connected to either a source or sink within the module or cell and are not connected to any other port on the interface of the module or cell.

The following also apply:

- It shall be an error if **-model** is specified and **-elements** is also specified.
- It shall be an error if -related_power_ports, -related_ground_ports, or -related_bias_ports is specified, but -model is not specified.
- It shall be an error if -related_ground_port is specified, but -related_power_port is not specified, or if -related_power_port is specified, but -related_ground_port is not specified.
- It shall be an error if -related_bias_port is specified, but either -related_power_port or -related_ground_port is not specified.
- It shall be an error if a supply port is included in **-ports** and that port has no pg type attribute.
- It shall be an error if **UPF_pg_type** is specified for a port that is not a supply port.
- It shall be an error if no argument is used.

Examples

```
set_port_attributes -ports {my_Logic_Port} -clamp_value 1
OR
set_port_attributes -ports {my_Logic_Port} -attribute {UPF_clamp_value "1"}
set_port_attributes -ports {my_Logic_Port}
-attribute {UPF_related_power_port "my_VDD"}
set_port_attributes -ports {my_Logic_Port}
-attribute {UPF_related_ground_port "my_VSS"}
set port attributes -ports {my_Logic_Port}
```

-attribute {UPF_related_bias_ports "my_VNWELL my_VPWELL "}

The following examples illustrate the use of **set_port_attributes** to specify user-defined attributes of ports, such as attribute values that might be required by tools or verification flows:

```
set_port_attributes -ports {a b c} -attribute {function {power nwell}}
set_port_attributes -ports {a} -attribute {voltage_range {0.0 1.2}}
set_port_attributes -ports {a -attribute {tester_control data}
```

6.47 set_power_switch [deprecated]

This is a deprecated command; see also 6.1 and Annex D.

6.48 set_repeater

Purpose	Specify a repeater (buffer) strategy		
Syntax	<pre>set_repeater strategy_name -domain domain_name [-elements element_list] [-exclude_elements exclude_list [-source <source_domain_name source_supply_ref="" ="">] [-sink <sink_domain_name sink_supply_ref="" ="">] [-use_equivalence [<true false="" ="">]] [-use_equivalence [<true both="" ="">] [-repeater_supply_set supply_set_ref] [-name_prefix string] [-name_suffix string] [-instance {{instance_name port_name}*}] [-update]</true></true></sink_domain_name></source_domain_name></pre>		
Arguments	strategy_name	The name of the repeater strategy.	
	-domain domain_name	The domain for which this strategy is defined.	
	-elements element_list	A list of instances or ports to which the strategy potentially applies.	R
	-exclude_elements exclude_list	A list of instances or ports to which the strategy does not apply.	R
	-source <source_domain_name source_supply_ref></source_domain_name 	The rooted name of a supply set or power domain. When a domain name is used, it represents the primary supply of the specified domain.	R
	-sink <sink_domain_name <br="">sink_supply_ref></sink_domain_name>	The rooted name of a supply set or power domain. When a domain name is used, it represents the primary supply of the specified domain.	R
	-use_equivalence [<true false="" ="">]</true>	Indicates whether to consider supply set equivalence. If -use_equivalence is not specified at all, the default is -use_equivalence TRUE ; if -use_equivalence is specified without a value, the default value is TRUE .	R
	-applies_to <inputs <br="">outputs both></inputs>	A filter that restricts the strategy to apply only to ports of a given direction.	R
	<pre>-repeater_supply_set supply_set_ref]</pre>	The supply set that powers the inserted buffer.	R

Arguments	-name_prefix string] [-name_suffix string]	The name format (prefix and suffix) for inserted buffer cell instances or nets related to implementation of the strategy.	R
	<pre>-instance {{instance_name port_name}*}</pre>	The name of a technology library leaf-cell instance and the name of the logic port that it buffers.	R
	-update	Indicates that this command provides additional information for a pre- vious command with the same <i>strategy_name</i> and <i>domain_name</i> and executed in the same scope.	R
Return value	Return an empty string if successful or raise a TCL_ERROR if not.		

The **set_repeater** command defines a strategy for inserting repeater cells (buffers) for ports on the interface of a power domain (see 6.17). Repeaters are placed within the domain, driven by input ports of the domain, and driving output ports of the domain.

-domain specifies the domain for which this strategy is defined.

-elements explicitly identifies a set of candidate ports to which this strategy potentially applies. The *element_list* may contain rooted names of instances or ports in the specified domain. If an instance name is specified in the *element_list*, it is equivalent to specifying all the ports of the instance in the *element_list*. Any *element_list* specified on the base command or any updates (see -update) of the base command are combined. If -elements is not specified in the base command or any update, every port on the interface of the domain is included in the *aggregate_element_list* (see 5.10).

-exclude_elements explicitly identifies a set of ports to which this strategy does not apply. The *exclude_list* may contain rooted names of instances or ports in the specified domain. If an instance name is specified in the *exclude_list*, it is equivalent to specifying all the ports of the instance in the *exclude_list*. Any *exclude_list* specified on the base command or any updates of the base command are combined into the *aggregate_exclude_list* (see <u>5.10</u>).

The arguments **-source**, **-sink**, and **-applies_to** serve as filters that further restrict the set of ports to which a given **set_repeater** command applies. The command only applies to those ports that satisfy all of the specified filters.

-source is satisfied by any port that is driven by logic powered by a supply set that matches (see -use_equivalence) the specified supply set, ignoring any isolation or level-shifting cells that have already been inferred or instantiated from an isolation or level-shifting strategy.

-sink is satisfied by any port that is received by logic powered by a supply set that matches (see -use_equivalence) the specified supply set, ignoring any isolation or level-shifting cells that have already been inferred or instantiated from an isolation or level-shifting strategy.

NOTE 1—A port that does not have a driver will never satisfy the **-source** filter. A port that does not have a receiver will never satisfy the **-sink** filter.

-use_equivalence specifies whether supply set equivalence is to be considered in determining when two supply sets match. If -use_equivalence is specified with the value *False*, the -source and -sink filters shall match only the named supply set. Otherwise, the -source and -sink filters shall match the named supply set or any supply set that is equivalent to the named supply set.

-applies_to is satisfied by any port that has the specified mode. For upper boundary ports, this filter is satisfied when the direction of the port matches. For lower boundary ports, this filter is satisfied when the

inverse of the direction of the port matches. For example, a lower boundary port with a direction OUT would satisfy the -applies_to IN filter, because an output from a lower boundary port is an input to this domain. -applies_to is always relative to the specified domain.

The *effective_element_list* (see <u>5.10</u>) for this command consists of all the port names in the *aggregate_element_list* that are not also in the *aggregate_exclude_list* and that satisfy all of the filters specified in the command. If a port in the *effective_element_list* is not on the interface of the specified domain, it shall not be buffered.

If a given port name is referenced in the *effective_element_list* of more than one repeater strategy of a given domain, the precedence rules (see <u>5.8</u>) determine which of those strategies actually apply to that port name. If the precedence rules identify multiple strategies that apply to the same port name, then the port name shall be the name of an input port to the domain, and each of those strategies shall each have a **-sink** filter that matches the receiving supply of a different sink domain for the specified input port. It shall be an error if the precedence rules identify multiple strategies that apply to the same port name and that port is an output port of the domain, or more than one strategy applies to the same sink domain for that port.

-repeater_supply_set is implicitly connected to the primary or backup supply ports of the buffer cell. If -repeater_supply_set is not specified, then if the primary supply set of the domain containing the driver of the repeater is available in the power domain where the repeater will be located, that supply is used as the default supply. It shall be an error if repeater_supply_set is not specified and the default supply is not available in the domain.

-name_prefix specifies the substring to place at the beginning of any generated name implementing this strategy.

-name_suffix specifies the substring to place at the end of any generated name implementing this strategy.

-instance specifies that the repeater functionality exists in the HDL design and *instance_name* denotes the instance providing part or all of this functionality. An *instance_name* is a simple name or a hierarchical name rooted in the current scope. If an empty string appears as an *instance_name*, this indicates that an instance was created and then optimized away. Such an instance should not be reinferred or reimplemented by subsequent tool runs.

-update adds information to the base command executed in the same scope. When specified with -update, -elements and -exclude_elements are additive: the set of instances or ports in the *aggregate_elements_list* is the union of all -elements specifications given in the base command and any update of this command, and the *aggregate_exclude_list* is the union of all -exclude_elements specifications given in the base command and any update of this command.

The following also apply:

- This command never applies to inout ports.
- The simstate semantics of the repeater supply set apply to the output of a repeater.

NOTE 2—To specify a repeater strategy for a port P on the lower boundary of a power domain D (see <u>4.3.1</u>), a **set_repeater** command can specify -domain D and specify the port name I/P, where I is the hierarchical name of an instance that is instantiated in domain D but is not in the extent of domain D, and P is the simple name of the port of that instance. The combination of the **-domain** specification and the hierarchical port name makes it clear this reference is to the HighConn of the specified port, which is part of the lower boundary of the domain D.

NOTE 3—Insertion of a repeater may change the driver supply and receiver supply of ports that are sinks or sources, respectively, of the inserted repeater. Such changes may affect the interpretation of **-source** or **-sink** filters of **set_isolation** (see <u>6.41</u>) or **set_level_shifter** (see <u>6.43</u>) strategies that apply to those ports. These changes may also affect the default for the input supply set or the output supply set of **set_level_shifter** strategies that apply to those ports.

NOTE 4—The *exclude_list* in **-exclude_elements** can specify instances or ports that have not already been explicitly or implicitly specified via an explicit or implied *element_list*.

Syntax example:

set_repeater feedthrough_buffer1
-domain PD3 -applies_to outputs

6.49 set_retention

Purpose	Specify a retention strategy		
Syntax	set_retention retention	name ne ist] [-exclude_elements exclude_list] set ret_supply_set] [-no_retention] net <high low="" negedge="" posedge="" ="">} ic_net <high low="" negedge="" posedge="" ="">}] solean_expression}] {boolean_expression}] {boolean_expression}] primary] r_SUP_COR NO_RET_SUP_COR NO_SAV_RES_COR> *}] ; [FALSE>]] c_name [signal_name]}*}] met net_name] [-retention_ground_net net_name]</high></high>	
	retention_name	Retention strategy name.	
	-domain domain_name	The domain for which this strategy is applied.	
	-elements element_list	The -elements option specifies a list of objects: instances, <i>retention_list_name</i> of elements lists (see 6.51), named processes, or sequential reg or signal names to which this strategy is applied.	R
	-exclude_elements exclude_list	The -exclude_elements option specifies a list of objects: instances, named processes, or sequential reg or signal names that are not included in this strategy.	R
Arguments	-no_retention	Prevents the inference of retention cells on the specified elements regardless of any other specifications.	R
	<pre>-retention_supply_set ret_supply_set</pre>	This option specifies the supply set used to power the logic inferred by the <i>retention_name</i> strategy.	R
	-save_signal {logic_net <high low="" posedge="" ="" <br="">negedge>} -restore_signal {logic_net <high low="" ="" <br="">posedge negedge>}</high></high>	The -save_signal and -restore_signal options specify a rooted name of a logic net or port and its active level or edge.	R
	-save_condition {boolean_expression}	The -save_condition option specifies a Boolean expression (see <u>5.4</u>). The default is <i>True</i> if the -save_signal/-restore_signals are specified, else the -save_condition is a don't care.	R

	-restore_condition {boolean_expression}	The -restore_condition option specifies a Boolean expression. The default is <i>True</i> if the -save_signal/-restore_signals are specified, else the -restore_condition is a don't care.	R
	-retention_condition {boolean_expression}	The -retention_condition option specifies a Boolean expression. The default is <i>True</i> if the -save_signal/-restore_signals are specified, else the default value of -retention_condition is <i>False</i> .	R
	-use_retention_as_pri mary	The -use_retention_as_primary option specifies that the storage element and its output are powered by the retention supply.	R
Arguments Legacy arguments	-parameters { <ret_sup_cor <br="">NO_RET_SUP_COR SAV_RES_COR NO_ SAV_RES_COR> *}</ret_sup_cor>	The -parameters option provides control over retention register corruption semantics.	R
	-transitive [<true <br="">FALSE>]</true>	If -transitive is not specified at all, the default is -transitive TRUE . If -transitive is specified without a value, the default value is TRUE .	
	<pre>-instance {{instance_name [signal_name]}*}</pre>	The name of a technology library leaf-cell instance and the optional name of the signal that it retains. If this instance has any unconnected supply ports or save and restore control ports, then these ports need to have identifying attributes in the cell model, and the ports shall be connected in accordance with this set_retention command.	R
	-update	Use -update if the <i>retention_name</i> has already been defined.	R
	<pre>-retention_power_net net_name</pre>	This option defines the supply net used as the power for the retention logic inferred by this strategy. This is a legacy option; see also 6.1 and <u>Annex D</u> .	R
	<pre>-retention_ground_net net_name</pre>	This option defines the supply net used as the ground for the retention logic inferred by this strategy. This is a legacy option; see also 6.1 and <u>Annex D</u> .	R
Return value	Return an empty string if	successful or raise a TCL_ERROR if not.	

The **set_retention** command specifies a set of objects in the domain that need to be retention registers and identifies the save and restore behavior. If an instance is specified, all registers within the instance acquire the specified retention strategy. If a process is specified, all registers inferred by the process acquire the specified retention strategy. If a reg, signal, or variable is specified and that object is a sequential element, the implied register acquires the specified retention strategy. Any specified reg, signal, or variable that does not infer a sequential element shall not be changed by this command.

If -elements is specified, only elements in the element list that are also a part of the *domain_name* are included. Any element names outside the extent of *domain_name* are excluded. When -elements is not specified, this is equivalent to using the elements list that defines the power domain. When used with -update, -elements is additive such that the set of elements or signals is the union of all calls of this command for a given strategy specifying any of these parameters.

-exclude_elements can also be used to define a list of storage elements that are not included in this strategy. When used with -update, -exclude_elements is additive such that the set of elements or signals excluded is the union of all calls of this command for a given strategy. **-retention_supply_set** powers the register holding the retained value. After the strategy has been completely applied, it shall be an error if the retention supply set is not defined for a strategy and the domain does not have a default *ret_supply_set*.

For a balloon-style retention register (see 4.3.4), the retained value is transferred to the register on the restore event when **-restore_condition** evaluates to *True*. The restore event is the rising or falling edge of an edge-triggered restore event or the trailing edge of a level-sensitive restore event. A level-sensitive restore event has priority over any other register operation.

-restore_condition gates the restore event, defining the restore behavior of the register. If the -save_signal/ restore_signals are not specified, the -restore_condition becomes a don't care. The register is restored when the restore event occurs and the -restore_condition is *True*.

For a balloon-style retention register, the retained value shall be the register's value at the time of the save event when **-save_condition** evaluates to *True*. The save event is the rising or falling edge of an edge-triggered save event or the trailing edge of a level-sensitive save event.

-save_condition gates the save event, defining the save behavior of the register. If the -save_signal/ restore_signals are not specified, the -save_condition becomes a don't care. The register contents are saved when the save event occurs and the -save_condition is *True*.

-retention_condition defines the retention behavior of the retention element while the primary supply is not NORMAL. If the retention condition evaluates to FALSE and the primary supply is not NORMAL, the receiving supply of any pin listed in the -retention_condition shall be assumed to be the retention supply of the retention strategy.

-save_condition, -restore_condition, and -retention_condition shall only reference logic nets or ports rooted in the current scope. The -save_signal/-restore_signal/-save_condition/-restore_condition apply only to balloon-style retention registers. For master-/slave-alive implementations (see 4.3.4), the -save_signal/-restore_signal should not be specified. The retention behavior of this style is specified through the -retention_condition. It shall be an error if -save_signal/-restore_signal is not specified and the -retention condition is also not specified.

-use_retention_as_primary powers the storage element and the output drivers of the register using the retention supply. The result of this is the simstate for the retention supply set is applied to the register's output. Inferred state elements shall be consistent with the **-use retention as primary** constraint.

NOTE 1—UPF only supports the output pins' driving supply being different from the primary supply (with **-use_retention_supply_as_primary**); the input pins' receiving supply can only be assumed to be the primary supply of the domain.

NOTE 2—The **-use_retention_as_primary** changes the driver supply of ports that are sinks of the inserted retention register. Such changes may affect the interpretation of the **-source** filters of the **set_repeater** (see <u>6.48</u>), **set_isolation** (see <u>6.41</u>), or **set_level_shifter** (see <u>6.43</u>) strategies that apply to those ports.

The **-parameters** option provides control over retention register corruption semantics. For a retention strategy, it is an error to specify:

- both **RET_SUP_COR** and **NO_RET_SUP_COR**; or
- both **SAV_RES_COR** and **NO_SAV_RES_COR**.

RET_SUP_COR activates and **NO_RET_SUP_COR** deactivates corruption of the normal mode register when retention supplies are **CORRUPT**. When neither value is specified for a retention strategy, **RET_SUP_COR** is the default value.

SAV_RES_COR activates and NO_SAV_RES_COR deactivates corruption of the normal mode register during concurrent assertion of level-sensitive save, save_condition, restore, and restore_condition. When neither value is specified for a retention strategy, SAV_RES_COR is the default value.

-instance specifies that the retention functionality exists in the HDL design and *instance_name* denotes the instance providing part or all of this functionality. An *instance_name* is a hierarchical name rooted in the current scope. If an empty string appears in an *instance_name*, this indicates that an instance was created and then optimized away. Such an instance should not be reinferred or reimplemented by subsequent tool runs.

-update adds information to the base command executed in the same scope of the power domain for which the inferred cells are defined.

The elements requiring retention can be attributed in HDL as shown in 6.51.

For details on the simulation semantics of this command, please refer to 9.6.

Examples

Some examples of the set_retention command are shown as follows:

a) Save-restore balloon-type RFF

Has an explicit save and restore pin, which perform save/restore functions.

```
set_retention my_ret \
-save_signal {save high} \
-restore_signal {restore high} \
...
```

- b) Single retention pin balloon-type RFF
 - 1) Has single pin that performs save/restore functions.
 - 2) To remain in a retention state, the retention pin shall be kept at a certain value.

```
set_retention my_ret \
-save_signal {ret posedge} \
-restore_signal {ret negedge} \
-retention_condition {ret} \
...
```

- c) Single retention pin slave-alive type RFF
 - 1) Has a single retention control pin, but no save/restore is involved as the slave latch (or storage element) is powered by the retention supply.
 - 2) Requires the retention pin to remain at a certain value to be in retention mode.

```
set_retention my_ret \
-retention_condition {ret} \
...
```

NOTE—No save/restore signals/conditions are specified in this case. Here, the retention condition is explicitly specified, meaning the retention condition has to be true during retention mode.

- d) No pin slave alive type RFF with output powered by retention supply
 - 1) Has no control pin and no save/restore is involved as the slave latch (or storage element) is powered by the retention supply.
 - 2) Requires the clocks/async resets to be related to retention supply and parked at a certain value during retention mode.
 - 3) The **-use_retention_as_primary** is specified as the output is expected to be powered by the retention supply.

```
set_retention my_ret \
-retention_condition {!clock && nreset} \
-use_retention_as_primary \
...
```

6.50 set_retention_control [deprecated]

This is a deprecated command; see also 6.1 and <u>Annex D</u>. To model mutex assertions using **bind_checker**, see 6.9.

6.51 set_retention_elements

Purpose	Create a named list of elements to be used in set_retention or map_retention_cell commands	
Syntax	set_retention_elements retention_list_name -elements element_list [-applies_to < required not_optional not_required optional>] [-exclude_elements exclude_list] [-retention_purpose < required optional>] [-transitive [<true false="" ="">]] [-expand [<true false="" ="">]]</true></true>	
	retention_list_name	A simple name; this shall be unique within the current scope.
Arguments	-elements element_list	A list of rooted names: instances, named processes, sequential $regs$, or signal names.
	-applies_to <required <br="">not_optional not_required optional></required>	Filter elements based on the UPF_retention attribute value.
	-exclude_elements exclude_list	A list of rooted names: instances, named processes, sequential $regs$, or signal names.
	-retention_purpose <required optional="" =""></required>	The intended retention use of <i>retention_list_name</i> . The default is required .
	-transitive [<true <br="">FALSE>]</true>	If -transitive is not specified at all, the default is -transitive TRUE . If -transitive is specified without a value, the default value is TRUE .
Deprecated arguments	-expand [<true <br="">FALSE>]</true>	When -expand is TRUE (the default), elements are expanded as though every register that otherwise would be included had been specified directly in <i>element_list</i> . This is a deprecated option; see also <u>6.1</u> and <u>Annex D</u> .
Return value	Return an empty string if	successful or raise a TCL_ERROR if not.

The set_retention_elements command defines a "atomic" list of objects whose state shall be retained or not retained together by the set_retention and map_retention_cell commands (see 6.49 and 6.33).

If the state of any element in *retention_element_list* is retained, the state of every element in *retention_element_list* shall be retained.
-applies_to filters the *effective_element_list*, removing any elements that do not have a UPF_retention attribute value consistent with the selected filter choice: required, not_optional, not_required, or optional, as follows:

required matches all elements that have the UPF_retention attribute value required.

optional matches all elements that have the UPF_retention attribute value optional.

not_required matches all elements that do not have the UPF_retention attribute value required.

not_optional matches all elements that do not have the UPF_retention attribute value optional.

When **-retention_purpose** is **required**, retention shall only be necessary if elements in the *retention element list* are in the extent of a power domain that has retained elements.

It shall be an error if an element belonging to *retention_element_list* is not retained when any element in the same *retention element list* is retained.

It shall be an error if **retention_purpose** is **required** and an element belonging to *retention_element_list* is not retained when any element in the same power domain extent is retained.

Syntax example:

```
set_retention_elements ret_chk_list
    -elements {proc_1 sig_a}
```

6.52 set_scope

Purpose	Specify the current scope	
Syntax	<pre>set_scope instance</pre>	
Arguments	instance	The instance that becomes the current scope upon completion of the command.
Return value	Return the current scope prior to execution of the command as a design-relative hierarchical name (see $5.3.3.3$) if successful or raise a TCL_ERROR if it fails (e.g., if the instance does not exist).	

The **set_scope** command changes the current scope to the specified scope and returns the name of the previous scope as a design-relative hierarchical name.

The following also apply:

- The instance name may be a simple name, a scope-relative hierarchical name, a design-relative hierarchical name, the symbol /, the symbol ...
- If the instance name is /, the current scope is set equal to the current design top instance.
- If the instance name is ., the current scope is unchanged.
- If the instance name is ..., and the current scope is not equal to the current design top instance, the current scope is changed to the parent scope.
- It is an error if the instance name is . . and the current scope is equal to the current design top instance.

Examples

Given the hierarchy

```
top/
mid/
bot/
```

if the current design top instance is /top, and the current scope is /top/mid, then

```
set_scope bot ;# changes current scope to /top/mid/bot (child of current scope)
set_scope . ;# leaves current scope unchanged as /top/mid (current scope)
set_scope .. ;# changes current scope to /top (parent of current scope)
set scope / ;# changes current scope to /top (current design top instance)
```

If the current design top instance is /top/mid and the current scope is /top/mid, then

```
set_scope bot ;# changes current scope to /top/mid/bot
set_scope . ;# leaves current scope unchanged as /top/mid
set_scope .. ;# results in an error
set scope / ;# changes current scope to /top/mid (current design top instance)
```

If the current design top instance is /top and the current scope is /top, then

```
set_scope mid/bot ;# changes current scope to /top/mid/bot
set_scope . ;# leaves current scope unchanged as /top
set_scope .. ;# results in an error
set scope / ;# changes current scope to /top (current design top instance)
```

6.53 set_simstate_behavior

Purpose	Specify the simulation simstate behavior for a model or library	
Syntax	<pre>set_simstate_behavior <enable disable="" =""> [-lib name] [-model model_list] [-elements element_list] [-exclude_elements exclude_list]</enable></pre>	
	<enable disable="" =""></enable>	Define if the UPF simstate behavior shall be enabled for the specified model(s).
Arguments	-lib name	The library name.
	-model model_list	One or more model names.
	-elements element_list	A list of instances.
	-exclude_elements exclude_list	A list of instances to exclude from the <i>effective_element_list</i> (see 5.10).
Return value	Return an empty string if successful or raise a TCL_ERROR if not.	

This command specifies the simstate behavior for models or instances.

If **ENABLE** is specified, the simulation semantics are applied for every supply set automatically connected to an instance of the model. See also <u>9.4</u>.

- a) If there is a single supply set connected, the simstates for that supply set are applied.
- b) When no supply set is connected, and each port to which a supply net is connected is of a different *pg_type*, an anonymous supply set is created containing the supply nets connected to each port, with each supply net associated with the function appropriate for the *pg_type* of that port, and the default simstates for that supply set are applied for the model.
- c) When there are multiple supply sets connected, the simstates of all supply sets are applied.
- d) For a hard macro instance in which there are multiple supply pins of the same pg_type, an anonymous supply set is created for each unique combination of supply pins identified as related supplies of a logic pin of the macro instance, with each supply pin associated with the function appropriate for the pg_type of that pin. The default simstates of each supply set are applied during simulation for any logic pin related to that supply set.
- e) For an instance of a hard macro behavioral model, each logic pin of the instance is corrupted according to the applicable simstate of the supply set associated with the logic pin.

If -model is not defined and -lib is specified, the simstate behavior is defined for all models in name.

It shall be an error if

- - **model** is specified and any of the model(s) cannot be found.
- -elements is specified and any of the element(s) cannot be found.
- **-exclude_elements** is specified and any of the *exclude_elements*(s) cannot be found.
- exclude_elements is specified and -model, -elements, or -lib is not specified.
- A given model has its simstate behavior both enabled and disabled, by set_simstate_behavior commands, UPF_simstate_behavior attributes, or a combination thereof.
- *effective_element_list* is empty.

Simstate behavior of a module can be enabled or disabled in HDL using the following attributes:

Attribute name: **UPF_simstate_behavior**

Attribute value: <"ENABLE" | "DISABLE">

SystemVerilog or Verilog-2005 example:

(* UPF simstate behavior = "ENABLE" *) module my adder;

VHDL example:

```
attribute UPF_simstate_behavior of my_adder : entity is
"ENABLE";
```

Syntax example:

set_simstate_behavior ENABLE -lib library1 -model ANDX7_non_power_aware

6.54 upf_version

Purpose	Retrieves the version of UPF being used to interpret UPF commands and documents the UPF version for which subsequent commands are written	
Syntax	<pre>upf_version [string]</pre>	
Arguments	string	The UPF version for which subsequent commands are written.

Return	Returns the version of UPF currently being used to interpret UPF commands.
value	

The **upf_version** command returns a string value representing the UPF version currently being used by the tool reading the UPF file. When the UPF version defined by this standard is being used, the returned value shall be the string "2.1". **upf_version** may also include an argument that documents the UPF version for which the UPF commands that follow were written. For UPF commands intended to be interpreted according to the UPF version defined by this standard, the argument shall be the string "2.1".

This standard does not define any other value for the returned value of the **upf_version** command or for the *string* argument. This standard also does not define how a tool uses the specified UPF version argument; in particular, this standard does not define the meaning of a description consisting of UPF commands intended to be interpreted according to different UPF versions.

Syntax example:

upf_version 2.1

6.55 use_interface_cell

Purpose	Specify the functional model and a list of implementation targets for isolation and level-shifting	
Syntax	<pre>use_interface_cell interface_implementation_name -strategy list_of_isolation_level_shifter_strategies -domain domain_name -lib_cells lib_cell_list [-port_map {{port net_ref}*}] [-elements element_list] [-exclude_elements exclude_list] [-applies_to_clamp <0 1 any Z latch value>] [-update_any <0 1 known Z latch value>] [-force_function] [-inverter_supply_set list]</pre>	
	interface_ implementation_name	The interface cell implementation strategy.
	-strategy list_of_isolation_level_ shifter_strategies	The isolation or level-shifter strategy, or a pair of isolation and level-shifter strategies, as defined by set_isolation and set_level_shifter .
	-domain domain_name	The domain in which the strategies are defined.
Arguments	-lib_cells lib_cell_list	A list of library cell names.
	<pre>-port_map {{port net_ref}*}</pre>	The <i>port</i> and the net (<i>net_ref</i>) connections.
	-elements element_list	A list of ports from the <i>list_of_isolation_level_shifter_strategies</i> to which the command applies.
	-exclude_elements exclude_list	A list of ports from the <i>list_of_isolation_level_shifter_strategies</i> to which this command does not apply.

Arguments	-applies_to_clamp <0 1 any Z latch value>	Only ports that have the specified clamp value are mapped.
	-update_any <0 1 known Z latch value>	What is now the clamp value when -applies_to_clamp is any .
	-force_function	The first model in <i>lib_cell_list</i> is used as the functional specification of isolation behavior.
	<pre>-inverter_supply_set list</pre>	The supply set implicitly connected to any inversion logic required by an isolation signal connection.
Return value	Return an empty string if	successful or raise a TCL_ERROR if not.

The use_interface_cell command provides user control for the integration of isolation and level-shifting. The command specifies the implementation choices through -lib_cells and the functional isolation behavior to be used if -force_function is specified.

Each cell specified in **-lib_cells** shall be defined by a **define_isolation_cell** (see <u>7.4</u>) or **define_level_shifter_cell** (see <u>7.5</u>) command or defined in the Liberty file with required attributes.

NOTE—Unlike **map_isolation_cell** and **map_level_shifter_cell**, **use_interface_cell** can be used to manually map any of isolation, level-shifting, or combined isolation level-shifting cells. It may apply to an isolation strategy, a level-shifting strategy, or one of each.

When **-force_function** is specified the first model in *lib_cell_list* shall be used as the functional model. The isolation sense specification for the isolation strategy is ignored when **-force_function** is specified. It is erroneous if the functional model clamps to a value that is different to the previously specified port clamp value.

-elements selects the ports from the specified list of strategies to which the mapping command is applied. If -elements is not specified, all ports inferred from the list of strategies shall have the mapping applied. When -applies_to_clamp is specified, this command is applied only to the ports with that clamp value.

When **-applies_to_clamp** is **any**, **-update_any** shall be used to specify the clamp value after mapping. An **-update_any** value of **known** specifies that the isolation function is more complex than can be specified by a single value.

-port_map connects the specified net ref to a port of the model. A net ref may be one of the following:

- a) A logic net name
- b) A supply net name
- c) One of the following symbolic references
 - 1) isolation_supply_set.function_name

function_name refers to the supply net corresponding to the function it provides to the **isolation_supply_set**.

- 2) **isolation_supply_set**[*index*].*function_name*
 - i) *index* is a non-negative integer corresponding to the position in the **isolation_supply_set** list specified for the isolation strategy.
 - ii) The **isolation_supply_set** *index* shall be specified if the isolation strategy specified more than one **isolation_supply_set**.

3) isolation_signal

- i) Refers to the isolation signal specified in the corresponding isolation strategy.
- ii) To invert the sense of the isolation signal the Verilog bit-wise negation operator ~ can be specified before the isolation_signal. The logic inferred by the negation shall be implicitly connected to the **inverter_supply_set** if specified, otherwise the **isolation_supply_set** shall be used.
- 4) isolation_signal[index]
 - i) *index* is a non-negative integer corresponding to the position in the **isolation_signal** list specified for the isolation strategy.
 - ii) The **isolation_signal** *index* shall be specified if the isolation strategy specified more than one **isolation_signal**.
 - iii) To invert the sense of the isolation signal the Verilog bit-wise negation operator ~ can be specified before the isolation_signal. If the isolation_signal is being inverted then the inverter_supply_set[index] if specified shall be implicitly connected to the inferred inverter, otherwise the isolation_supply_set[index] shall be used.
- 5) **input_supply_set**.function_name

function_name refers to the supply net corresponding to the function it provides to the level-shifter **input_supply_set**.

6) **output_supply_set**.function_name

function_name refers to the supply net corresponding to the function it provides to the level-shifter **output_supply_set**.

7) internal_supply_set.function_name

function_name refers to the supply net corresponding to the function it provides to the level-shifter **internal_supply_set**.

The **-port_map** option shall not reference the data input port or the data output port. The input port shall be connected to the data input for the interface cell and the output port connected to the data output for the interface cell.

It shall be an error if

- *domain_name* does not indicate a previously created power domain.
- A port in the *port_list* is not covered by a **set_isolation** command.
- *list_of_isolation_level_shifter_strategies* is an empty list.
- -- -force_function is not specified and none of the specified models in *lib_cell_list* implements the functionality specified by the corresponding *isolation_strategy* and port attributes.
- - **update_any** is specified and **-applies_to_clamp** is not **any**.
- After completing the *port* and *net_ref* connections and the data input and output connections, any port is unconnected.
- Ports specified by -elements are not included in all specified strategies.
- More than one isolation strategy is specified.
- More than one level-shifter strategy is specified.

Syntax example:

```
use_interface_cell my_interface -strategy {ISO1 LS1} -domain PD1\
-elements {top/moduleA/port1 top/moduleA/port2 top/moduleA/port3}
```

7. Power management cell commands

7.1 Introduction

This clause documents the syntax for each UPF power management cell command. A power management cell is one of the following:

- "Always-on" cell
- Diode clamp
- Isolation cell
- Level-shifter cell
- Power-switch cell
- Retention cell

Power management cell commands define characteristics of the instances of power management cells used to implement and verify the power intent for a given design. These commands do not alter the existing library cell definitions and only have semantics when they are used with design power intent commands (see <u>Clause 6</u>).

Similar to how libraries are processed in a design flow, UPF power management cell commands need to be processed before any other power intent commands and after the relevant cell libraries have been loaded.

It is an error if conflicting information is specified in multiple commands (of any type).

To understand the relationship between each UPF power management cell command and its library cell definition in Liberty format, see <u>Annex H</u>.

7.2 define_always_on_cell

Purpose	Identify always-on cells	
Syntax	define_always_on_cell -cells cell_list -power pin -ground pin [-power_switchable pin] [-ground_switchable pin] [-isolated_pins list_of_pin_lists [-enable expression_list]]	
	-cells cell_list	Identifies the specified cells as always-on cells.
Arguments	-power pin	Identifies the power pin of the cell. If this option is specified with the -power_switchable option, it indicates this is a non-switchable power pin.
	-ground pin	Identifies the ground pin of the cell. If this option is specified with the -ground_switchable option, it indicates this is a non-switchable ground pin.
	-power_switchable pin	Specifies the power pin to be connected via a rail connection to the switchable power supply.
	-ground_switchable pin	Specifies the ground pin to be connected via a rail connection to the switchable ground supply.
	-isolated_pins list_of_pin_lists	Specifies a list of pin lists. Each pin list groups pins that are isolated internally with the same isolation control signal. These pin lists can only contain input pins.
	-enable expression_list	Specifies a list of simple expressions. Each simple expression describes the isolation control condition for the corresponding isolated pin list in the -isolated_pins option. If the internal isolation does not require a control signal, use an empty string for that pin list. The number of elements in this list shall correspond to the number of lists specified for the -isolated_pins option.
Return value	Return an empty string if	successful or raise a TCL_ERROR if not.

The **define_always_on_cell** library command identifies the library cells having more than one set of power and ground pins that can remain functional even when the supply to the switchable power or ground pin is switched off.

NOTE—Although a cell is called *always-on* does not mean the cell can never be powered off. When the supply to non-switchable power or ground of such cell is switched off, the cell becomes non-functional. In other words, the term always-on actually means relative always-on.

By default, all input and output pins of this cell are related to the non-switchable power and ground pins.

Examples

The following example defines cell aon_cell as an always-on cell. The cell had three isolated pins: pin1, pin2, and pin3. Pins pin1 and pin2 have the same isolation control signal iso1, but pin3 has no isolation control signal.

```
define_always_on_cell -cells aon_cell
    -isolated pins { {pin1 pin2} {pin3}} -enable {!iso1 ""}
```

The following example defines cell AND2_AON as an always-on cell. The cell has two power pins and performs the AND function (as defined in the library) as long as the supply connected to power pin VDD is not switched off.

define_always_on_cell -cells AND2_AON -power_switchable VDDSW
 -power VDD -ground VSS

7.3 define_diode_clamp

Purpose	Identify diode cells or cells pins with diode protection	
Syntax	define_diode_clamp -cells cell_list -data_pins pin_list [-type <power both="" ground="" ="">] [-power pin] [-ground pin]</power>	
	-cells cell_list	Identifies cells as diode clamp cells or pins of the specified cells as diode clamp pins.
Arguments	<pre>-data_pins pin_list</pre>	Specifies a list of cell input pins that have built-in clamp diodes.
	-type <power ground="" ="" <br="">both></power>	Specifies the type of clamp diode associated with the data pins. The type determines whether to use the power pin, ground pin, or both. Possible values are as follows: both indicates a power and ground clamp diode ground indicates a ground clamp diode power indicates a power clamp diode (the default)
	-power pin	Specifies the cell pin that connects to the power net.
	-ground pin	Specifies the cell pin that connects to the ground net.
Return value	Return an empty string if	successful or raise a TCL_ERROR if not.

The **define_diode_clamp** library command identifies a list of library cells that are power cells, ground cells, or power and ground diode clamp cells, or complex cells that have input pins with built-in clamp diodes.

When **-type** is **ground**, then **-power** is optional. When **-type** is power, then **-ground** is optional. When **-type** is **both**, then both **-power** and **-ground** need to be specified as well.

It shall be an error if neither **-power** nor **-ground** is specified.

NOTE—The **define_diode_clamp** command is typically used for pins that have antenna protection diodes. Hence, this command may apply to regular non-power managed cells.

Examples

The following command defines a cell cellA with diode protection at the pin in1 where the diode is connected to the power pin VDD1 of the cell.

define_diode_clamp -cells cellA -data_pins in1 -type power -power VDD1

7.4 define_isolation_cell

Purpose	Identify isolation cells	
Syntax	define_isolation_cell -cells cell_list [-power power_pin] [-ground power_pin] {-enable pin [-clamp_cell < high low>] -pin_groups {{input_pin output_pin [enable_pin]}*} -no_enable < high low hold>} [-always_on_pins pin_list] [-aux_enables ordered_pin_list] [-power_switchable power_pin] [-ground_switchable ground_pin] [-valid_location <source any="" off="" on="" sink="" =""/>] [-non_dedicated]	
	-cells cell_list	Identifies the specified cells as isolation cells.
Arguments	-power power_pin	Identifies the power pin of the cell. If this option is specified with the -power_switchable option for a multi- rail isolation cell, it indicates this is a non-switchable power pin.
	-ground power_pin	Identifies the ground pin of the cell. If this option is specified with the -ground_switchable option for a multi- rail isolation cell, it indicates this is a non-switchable ground pin.
	-enable pin	Identifies the specified cell pin as the isolation enable pin. For non-clamp type isolation cells, the enable pin polarity is determined by the cell function defined in the library files. For the special clamp type cell identified by the -clamp_cell option, the enable polarity is active high if the clamp output is low and the enable polarity is active low if the clamp output is high. For a multi-rail isolation cell, the enable pin is related to the non-switchable power and ground pins of the cells.
	-clamp_cell <high <br="">low></high>	Indicates the specified cells are isolation clamp cells. Such a cell, which consists of a single PMOS or NMOS transistor, does not perform any logic function and is only used to clamp a net to high or low when the enable pin is activated.
	<pre>-pin_groups {{input_pin output_pin [enable_pin]}*}</pre>	Specifies a list of input-output paths for multi-bit isolation cells. Each group in the list specifies one cell input pin, one cell output pin, and one optional enable pin that applies to the specified path. An enable pin may appear in more than one group. It is an error if the same input or output pin appears in more than one group.
	-no_enable <high low="" ="" <br="">hold></high>	Specifies the following: The isolation cell does not have an enable pin. The output of the cell when the supply for the switchable power (or ground) pin is powered down. Possible values are as follows: high indicates the cell output is logic value 1 low indicates the cell output is logic value 0 hold indicates the cell output is the same as the logic value before the supply for the switchable power or ground is powered down
	-always_on_pins pin_list	Specifies a list of cell pins related to the nonswitchable power and non- switchable ground pins of the cell.
	-aux_enables ordered_pin_list	Specifies additional or auxiliary enable pins for the isolation cell.

Arguments	-power_switchable power_pin	Identifies the switchable power pin of a multi-rail isolation cell.
	-ground_switchable ground_pin	Identifies the switchable ground pin of a multi-rail isolation cell.
	-valid_location <source sink on off any></source 	Specifies the valid location of the isolation cell. The default value is sink .
	-non_dedicated	Allows the specified cells to be used as normal cells, not for power management purposes.
Return value	Return an empty string if successful or raise a TCL_ERROR if not.	

The **define_isolation_cell** library command identifies the library cells that can be used for isolation in a design with power gating.

By default, the output pin of a multi-rail isolation cell is related to the non-switchable power and ground pins. The non-enable input pin is related to the switchable power and ground pins. A *multi-rail isolation cell* is a cell with two power or ground pins.

If -clamp_cell is specified with value high, the only supply pin that can be specified is -power. If -clamp_cell is specified with low, the only supply pin that can be specified is -ground. For all other isolation cells, both -power and -ground shall be specified.

The **-aux_enables** option specifies additional or auxiliary enable pins for the isolation cell. By default, all pins specified in this option are related to the switchable power or ground pin. The list is an ordered list and each element can be accessed by using index starting at 1, where the isolation enable pin specified in the **-enable** option is assumed to be index 0.

If an auxiliary enable pin is related to the non-switchable power or ground, that pin shall also be specified using the **-always_on_pins** option. The logic that drives this pin shall be on when the isolation enable is asserted at pin specified by the **-enable** option.

The **-valid_location** option specifies the valid location of the isolation cell, as follows:

- a) **source**—indicates the cell shall be inserted in a location where the primary supply set is equivalent to the driving supply set for a net requiring isolation. Such cells are typically multi-rail isolation cells and used for off-to-on isolation. It typically relies on its switchable power and ground supply for its normal function and on its non-switchable power or ground supply to provide the isolation function. See item d) for **off** value for special cases.
- b) **sink**—indicates the cell shall be inserted in a location where the primary supply set is equivalent to the receiving supply set for a net requiring isolation. Such cells are typically single-rail isolation cells and used for off-to-on isolation.
- c) **on**—indicates the cell can only be inserted in the location where the primary supply set is equivalent to either the driving supply set or the receiving supply for a net requiring isolation and the primary supply set is not switched off when the isolation function is needed. When used for off-to-on isolation, it is equivalent to **sink**. Such cells are typically single-rail isolation cells.
- d) **off**—indicates the cell can be inserted in a location where the primary supply set is equivalent to either the driving supply set or the receiving supply for a net requiring isolation and the primary supply set may be switched off when the isolation function is needed. When used for off-to-on isolation, it is equivalent to **source**. Such cells are typically multi-rail isolation cells.

NOTE—Some single-rail isolation cells with special circuit structure can also be used in the switched-off domain. For example, a single-rail NOR gate can be placed in a power-switched-off domain for off-to-on isolation with an output value low; a single-rail NAND gate can be placed in the ground switched-off domain for off-to-on isolation with an output value high.

e) **any**—indicates the cell can be placed in any location. Such cells are typically multi-rail isolation cells. In addition, this cell is designed in a way that neither its normal function nor its isolation function relies on the primary supply of the domain it locates. Therefore, this type of cell can be used for off-to-on or on-to-off isolation.

Examples

The following isolation cell can be placed in any location for a design that uses ground switches for shutoff. VDD is the rail pin for power connection and GVSS is the ground pin for non-switchable ground connection. This cell does not have a rail pin for ground connection.

```
define_isolation_cell -cells iso_cell1 -power VDD -ground GVSS
     -enable iso_en -valid_location any
```

The following examples illustrate the use of the **-pin_groups** option to specify multi-bit isolation cells with two paths:

```
define_isolation_cell -cells mbit_isol -pin_groups { { datain1 dataout1
    isol } { datain2 dataout2 iso2 } }
    -power VDD -ground VSS -valid_location sink
define_isolation_cell -cells mbit_iso2 -pin_groups { { datain1 dataout1 }
    { datain2 dataout2} }
    -power VDD -ground VSS -valid location sink
```

For cell mbit_iso1, there are two isolation paths. The first is from data input datain1 to output dataout1 with iso1 as the isolation enabler. The second is from data input datain2 to output dataout2 with iso2 as the isolation enabler.

For cell mbit_iso2, there are also two isolation paths. However, this special isolation cell has no isolation enabler to control each path. As a result, there is no isolation enable signal defined in each group.

7.5 define_level_shifter_cell

Purpose	Identify level-shifter cells	
Syntax	define_level_shifter_cell -cells cell_list [-input_voltage_range {{lower_bound upper_bound}*}] [-output_voltage_range {{lower_bound upper_bound}*}] [-ground_input_voltage_range {{lower_bound upper_bound}*}] [-ground_output_voltage_range {{lower_bound upper_bound}*}] [-ground_output_voltage_range {{lower_bound upper_bound}*}] [-ground_output_voltage_range {{lower_bound upper_bound}*}] [-ground_output_voltage_range {{lower_bound upper_bound}*}] [-direction <low_to_high both="" high_to_low="" ="">] [-input_power_pin power_pin] [-output_ground_pin ground_pin] [-output_ground_pin ground_pin] [-ground ground_pin] [-power power_pin] [-enable pin -pin_groups {{input_pin output_pin [enable_pin]}*}] [-valid_location <source any="" either="" sink="" =""/>] [-bypass_enable expression] [-multi_stage integer]</low_to_high>	
	-cells cell_list	Identifies the specified cells as level-shifter cells.
Arguments	<pre>input_voltage_range {{lower_bound upper_bound}*}</pre>	Identifies a list of voltage ranges for the input (source) supply voltage that can be handled by this level-shifter. The voltage range shall be specified as follows: {lower_bound upper_bound} This option should only be specified for power-shifting cells.
	<pre>-output_voltage_range {{lower_bound upper_bound}*}</pre>	Identifies a list of voltage ranges for the output (destination) power supply voltage that can be handled by this level-shifter. The voltage range shall be specified as follows: {lower_bound upper_bound} This option should only be specified for power-shifting cells.
	<pre>-ground_input_voltage _range {{lower_bound upper_bound}*}</pre>	Identifies a list of voltage ranges for the input (source) ground supply volt- age that can be handled by this level-shifter. The voltage range shall be specified as follows: {lower_bound upper_bound} This option should only be specified for ground-shifting cells.
	-ground_output _voltage_range {{lower_bound upper_bound}*}	Identifies a list of voltage ranges for the output (destination) ground supply voltage that can be handled by this level-shifter. The voltage range shall be specified as follows: {lower_bound upper_bound} This option should only be specified for ground-shifting cells.
	-direction <low_to_high <br="">high_to_low both></low_to_high>	Specifies whether the level-shifter can be used between a driver with lower voltage swing and a receiver with higher voltage swing (low_to_high), or vice versa (high_to_low), or both (both). The <i>voltage swing</i> is simply the difference between the power voltage and ground voltage. The default is low_to_high.
	-input_power_pin power_pin	Identifies the input power pin. This option is usually specified for power shifting and used with -output_power_pin .

	-output_power_pin power_pin	Identifies the output power pin. This option is usually specified for ground shifting and used with _input_power_pin .
	-input_ground_pin ground_pin	Identifies the input ground pin. This option is usually specified for ground shifting and used with -output_ground_pin .
	-output_ground_pin ground_pin	Identifies the output ground pin. This option is usually specified for ground shifting and used with _input_ground_pin .
	-ground ground_pin	Identifies the ground pin of the cell. This option can only be specified for level-shifters that only perform power shifting. In other words, it is an error to use this option with _input_ground_pin and _output_ground_pin .
Arguments	-power power_pin	Identifies the power pin of the cell. This option can only be specified for level-shifters that only perform ground shifting. In other words, it is an error to use this option with _input_power_pin and _output_power_pin .
	-enable pin	Identifies the pin that prevents internal floating when the power supply of the originating power domain is powered down, but the output voltage level power pin remains on. The related power and ground of this pin is the output power and ground pins defined for this cell.
	<pre>-pin_groups {{input_pin output_pin [enable_pin]}*}</pre>	Specifies a list of input-output paths for multi-bit isolation cells. Each group in the list specifies one cell input pin, one cell output pin, and one optional enable pin that applies to the specified path. An enable pin may appear in more than one group. It is an error if the same input or output pin appears in more than one group.
	-valid_location <source sink either any></source 	Specifies the valid location of the level-shifter cell. The default value is sink .
	-bypass_enable expression	Specifies when to bypass the voltage shifting functionality. When the expression evaluates to <i>True</i> , the cell behaves like a buffer. The expression shall be a simple expression of the bypass enable input pin. By default, the related power and ground of this pin is the output power and ground pin defined for this cell.
	-multi_stage integer	Identifies the stage of a multi-stage level-shifter to which this definition (command) applies. For a level-shifter cell with N stages, N definitions shall be specified for the same cell. Each definition needs to associate a number from 1 to N for this option. For more information, see <u>Annex I</u> .
Return value	Return an empty string if	successful or raise a TCL_ERROR if not.

The **define_level_shifter_cell** library command identifies the library cells to use as level-shifter cells, as follows:

- If -input_voltage_range is specified, the -output_voltage_range shall also be specified.
- If **-ground_input_range** is specified, the **-ground_output_range** shall also be specified.
- It is an error if neither -input_voltage_range nor -ground_input_voltage_range is specified.

If a list of voltages ranges is specified for the input supply voltage, a list of voltages ranges for the output supply voltage with the same number of elements shall also be specified., i.e., each member in the list of input voltage ranges needs to have a corresponding member in the list of output voltage ranges.

By default, the enable and output pins of this cell are related to the output power and output ground pins (specified through the **-output_power_pin** and **-output_ground_pin** options). And the non-enable input pin is related to the input power and input ground pins (specified through the **-input_power_pin** and **-input_ground_pin** options).

The -valid_location option specifies the valid location of the level-shifter cell, as follows:

- a) **source**—indicates the cell shall be inserted in a location where the primary supply set is equivalent to the driving supply set for a net requiring level-shifting.
- b) **sink**—indicates the cell shall be inserted in a location where the primary supply set is equivalent to the receiving supply set for a net requiring level-shifting.
- c) **either**—indicates the cell shall be inserted in a location where the primary supply set is equivalent to the driving supply set or the receiving supply set for a net requiring level-shifting.
- d) **any**—indicates the cell can be placed in any location.
 - 1) If the cell contains pins for rail connection, these pins shall not be specified through the **-input_power_pin**, **-output_power_pin**, **-input_ground_pin**, or **-output_ground_pin** options.
 - 2) A power level-shifter with this setting can be placed in any location as long as its primary ground net is equivalent to the driving and receiving primary ground net of the net requiring level-shifting.
 - 3) A ground level-shifter with this setting can be placed in any location as long as its primary power net is equivalent to the driving and receiving primary power net of the net requiring level-shifting.
 - 4) For a power and ground level-shifter, which requires two definitions of the command—one for the power part and one for the ground part of the cell—the **-valid_location** can be different in the two definitions.

Power part	Ground part
any	source sink either
source sink either	any
any	any

- i) In the first case, the ground-shifting part of the level-shifter definition determines the location.
- ii) In the second case, the power-shifting part of the level-shifter definition determines the location.
- iii) In the third case, the cell can be placed in a domain whose power and ground supplies are neither driving the logic power and ground supplies nor receiving the logic power and ground supplies.

Examples

The following example identifies level-shifter cells with one power pin and one ground pin that perform power shifting from 1.0 V to 0.8 V.

```
define_level_shifter_cell
-cells LSHL
-input_voltage_range {{1.0 1.0}} -output_voltage_range {{0.8 0.8}}
-direction high_to_low
-input_power_pin VH -ground G
```

The following example identifies level-shifter cells that perform power shifting from 0.8 V to 1.0 V. In this case, the level-shifter cells need to have two power pins and one ground pin.

```
define_level_shifter_cell
-cells LSLH
-input_voltage_range {{0.8 0.8}} -output_voltage_range {{1.0 1.0}}
-direction low_to_high
-input_power_pin VL -output_power_pin VH -ground G
```

The following example identifies level-shifter cells with valid location any to perform voltage shifting from 0.8 V to 1.0 V. The cells have three power pins and one ground pin.

VDD—This is the standard cell rail; this pin is not used by the cell.

VDDL—This is the power pin to which the input signal is related.

VDDH—This is the power pin to which the output signal is related.

VSS—This is the ground pin of the cell.

```
define_level_shifter_cell
-cells LSLH
-direction low_to_high
-input_voltage_range {{0.8 0.8}} -output_voltage_range {{1.0 1.0}}
-input_power_pin VDDL -output_power_pin VDDH -ground VSS
-valid location any
```

The following example identifies level-shifter cells that perform both power shifting from 0.8 V to 1.0 V and ground shifting from 0.2 V to 0 V. In this case, the level-shifter cells need to have two power pins and two ground pins. In addition, since the input voltage swing is 0.6 V (0.8 V – 0.2 V), which is smaller than the output voltage swing of 1.0 V (1.0 V - 0 V), the direction of the cell is low to high.

```
define_level_shifter_cell
-cells LSLH
-input_voltage_range {{0.8 0.8}} -output_voltage_range {{1.0 1.0}}
-ground_input_voltage_range {{0.2 0.2}} -ground_output_voltage_range {{0.0 0.0}}
-direction low_to_high
-input_power_pin VL -output_power_pin VH
-input_ground_pin GH -output_ground_pin GL
```

The following example indicates the level-shifter can shift from 0.8 V to 1.0 V or from 1.0 V to 1.2 V. However, the cell cannot shift power voltage from 0.8 V to 1.2 V.

```
define_level_shifter_cell
-cells LSLH
-input_voltage_range {{0.8 1.0}} -output_voltage_range {{1.0 1.2}}
-input_power_pin VL -output_power_pin VH -ground_pin VSS
-direction low_to_high
```

The following example indicates the level-shifter can shift from input range 0.8 V to 0.9 V to output range 1.0 V to 1.1 V, or from input range 0.9 V to 1.0 V to output range 1.1 V to 1.2 V. Note that the cell cannot shift input voltages between 0.8 V to 0.9 V to output voltages 1.1 V to 1.2 V.

```
define_level_shifter_cell
-cells LSLH -input_power_pin VL -output_power_pin VH -ground_pin VSS
-input_voltage_range {{0.8 0.9} {0.9 1.0}}
-output_voltage_range {{1.0 1.1} {1.1 1.2}}
-direction low_to_high
```

The following examples illustrate the use of the **-pin_groups** option to specify multi-bit level-shifter cells with and without enable:

```
define_level_shifter_cell -cells mbit_en_ls -pin_groups { { datain1
  els_dataout1 en1 } {datain2 els_dataout2 en2 } }
define_level_shifter_cell -cells mbit_ls -pin_groups { { datain1
  ls_dataout1 } { datain2 ls_dataout2 } }
```

7.6 define_power_switch_cell

Purpose	Identify a power switch or ground-switch cell	
Syntax	<pre>define_power_switch_cell -cells cell_list -type <footer header="" =""> -stage_1_enable expression [-stage_1_output expression] {-power_switchable power_pin -power power_pin -ground_switchable ground_pin -ground ground_pin]} [-stage_2_enable expression [-stage_2_output expression]] [-always_on_pins ordered_pin_list] [-gate_bias_pin power_pin]</footer></pre>	
	-cells cell_list	Identifies the specified cells as power-switch cells.
	-type <footer header="" =""></footer>	Specifies whether the power-switch cell is a header or footer cell.
	-stage_1_enable (-stage_2_enable) expression	Specifies when the switch cell driven by this input pin is turned on (enabled) or off. If only stage 1 is specified, the switch is turned on when the expression for the -stage_1_enable option evaluates to <i>True</i> and the switch is turned off when the expression for the -stage_1_enable option evaluates to <i>False</i> . If both stages are specified, the switch is turned on when the expression for both enable options evaluates to <i>True</i> and the switch is turned off when the expression for both enable options evaluates to <i>True</i> . The Boolean expression is a simple expression of the input pin.
Arguments	-stage_1_output (-stage_2_output) expression	Specifies whether the output pin in the expression is the buffered or inverted output of the input pin specified through the corresponding -stage_x_enable option. In a design, this pin is used to connect another switch cell in series to form a power-switch chain.
	-power_switchable power_pin	Identifies the output power pin in the corresponding cell. This option can only be used if the power gating cell is used to cut off the path from power to ground on the power side (i.e., for a header cell). This pin shall be connected to a switchable power net.
	<pre>-power power_pin</pre>	Identifies the input power pin of the cell.
	-ground_switchable ground_pin	Identifies the output ground pin in the corresponding cell. This option can only be used if the power gating cell is used to cut off the path from power to ground on the ground side (i.e., for a footer cell). This pin shall be connected to a switchable ground net.

Arguments	-ground power_pin	Identifies the input ground pin of the cell.
	-always_on_pins ordered_pin_list	Specifies a list of cell pins related to the input power and ground pins of the cell.
	-gate_bias_pin power_pin]	Identifies a power pin that provides the supply used to drive the gate input of the switch cell.
Return value	Return an empty string if successful or raise a TCL_ERROR if not.	

The **define_power_switch_cell** library command identifies the library cells to use as power-switch cells. The input enable and output enable pins of these cells are related to the non-switchable power and ground pins.

Examples

The following example defines a header power switch. The power switch has two stages. The power switch is completely on if the transistors of both stages are on. The stage 1 transistor is turned on by applying a low value to input I1. The output of the stage 1 transistor, O1, is a buffered output of input I1. The stage 2 transistor is turned on by applying a high value to input I2. The output of stage 2 transistor, O2, is the inverted value of input I2.

define_power_switch_cell -cells 2stage_switch -stage_1_enable !I1
-stage_1_output 01 -stage_2_enable I2 -stage_2_output !02 -type header

7.7 define_retention_cell

Purpose	Identify state retention cells	
Syntax	<pre>define_retention_cell -cells cell_list -power power_pin -ground ground_pin [-cell_type string] [-always_on_pins pin_list] [-restore_function {{pin < high low posedge negedge}}] [-save_function {{pin < high low posedge negedge}}] [-restore_check expression] [-save_check expression] [-retention_check expression] [-hold_check pin_list] [-always_on_components component_list] [-power_switchable power_pin] [-ground_switchable ground_pin]</pre>	
	-cells cell_list	Identifies the specified cells as state retention cells.
	-power power_pin	Identifies the power pin of the cell. If this option is specified with the -power_switchable option, it indicates this is a non-switchable power pin.
	-ground ground_pin	Identifies the ground pin of the cell. If this option is specified with the -ground_switchable option, it indicates this is a non-switchable ground pin.
	<pre>-cell_type string</pre>	Specifies a user-defined name grouping the specified cells into a class of retention cells that all have the same retention behavior. This specification limits the group of cells that can be used to those requested through the -lib_cell_type option of the map_retention_cell command (see <u>6.33</u>).
	-always_on_pins pin_list	Specifies a list of cell pins that are related to the nonswitchable power and ground pins of the cells.
Arguments	-restore_function {{pin <high low="" posedge="" ="" <br="">negedge}}</high>	Specifies the polarity or the edge sensitivity of the restore pin that enables the retention cell to restore the saved value after exiting power shut-off mode. By default, the restore pin relates to the non-switchable power and ground pin of the cell. If not specified, the restore event is triggered when the primary power is restored, or the power-up event. When neither -save_function nor -restore_function is specified, the current value is always saved before entering retention mode and the saved value is restored when the primary power is restored.
	-save_function {{pin <high low="" posedge="" ="" <br="">negedge}}</high>	Specifies the polarity or the edge sensitivity of the save pin that enables the retention cell to save the current value before entering retention mode. By default, the save pin relates to the non-switchable power and ground pin of the cell. If not specified, the save event is triggered by the negation of the restore function when it is specified. When neither -save_function nor -restore_function is specified, the current value is always saved before entering retention mode and the saved value is restored when the primary power is restored.
	-restore_check expression	Specifies the additional condition when the states of the sequential elements can be restored. The expression shall be a function of the cell input pins. The expression shall be <i>True</i> when the restore event occurs.

Arguments	-save_check expression	Specifies the additional condition when the states of the sequential ele- ments can be saved. The expression shall be a function of the cell input pins. The expression shall be <i>True</i> when the save event occurs.
	-retention_check expression	Specifies an additional condition to meet (after the primary supply of the retention cell is switched off and before the supply is powered on again) for the retention operation to be successful. <i>expression</i> can be a Boolean function of cell input pins. The <i>expression</i> shall be <i>True</i> when the primary supply set of the power domain, in which the retention logic locates, is shut off and the retention supply set is on.
	<pre>-hold_check pin_list</pre>	Specifies a list of pins that maintain the same logic value during the reten- tion period, from the time when the save event occurs to the time when the restore event occurs. The pin may be the clock pin or any other control pin.
	-always_on_ components component_list	Specifies a list of component names: instances, named processes, sequential reg, or signal names, in the corresponding simulation model that are powered by the nonswitchable power and ground pins. The logic values of the specified components are corrupted if the state value of the non-switchable power and group pin is OFF.
		NOTE—The option has only an impact on tools that use the gate-level simulation models of state retention cells.
	-power_switchable power_pin	Identifies the switchable ground pin. This cell can be used for retention purpose in a power domain that can be shutoff using power switches (i.e., using a header cell).
	-ground_switchable ground_pin	Identifies the switchable ground pin. This cell can be used for retention purpose in a power domain that can be shutoff using ground switches (i.e., using a footer cell).
Return value	Return an empty string if successful or raise a TCL_ERROR if not.	

The **define_retention_cell** library command identifies the library cells to use as retention cells. The following also apply:

- By default, all pins of this cell are related to the switchable power and ground pins, unless otherwise specified.
- It is an error if the save and restore functions both identify the same pin, and the polarity or edge sensitivity are the same for that pin. For example, the following two commands are incorrect:

```
define_retention_cell -cells My_Ret_Cell1
    -restore_function {pg high} -save_function {pg high}
define_retention_cell -cells My_Ret_Cell2
    -restore function {pg posedge} -save function {pg posedge}
```

— It is an error if the conditions specified in -save_check, -restore_check, or -retention_check conflict with -hold_check. For example, the specification

'-hold_check clk -save_check !clk -restore_check clk'

is an error since the -hold_check requires the clk signal to hold the same value from the time when the save event occurs to the time when the restore event occurs, but the other two options require the signal clk have different values.

NOTE—If the cell data output pin is listed in the **-always_on_pins** list, then this retention cell may be used for retention strategies that specify **-use_retention_as_primary**.

Example

In the following example, the cell design requires clock clk be held to 0 to save or restore the state of the sequential element. If retention control pin save is set to 0, the state will be saved and saved data will be restored when the primary power VDD is restored. The retention power VDDC shall be on to enable the retention while VDD is switched off.

```
define_retention_cell -cells My_Ret_Cell -power VDDC
-ground VSS -power_switchable VDD
-save_check {!clk} -restore_check {!clk}
-save_function {save negedge}
```

8. UPF processing

8.1 Overview

All UPF commands have an immediate effect when they are executed by a Tcl interpreter. For the following commands, the immediate effect is the only effect:

- create_hdl2upf_vct (see 6.14)
- create_upf2hdl_vct (see <u>6.23</u>)
- find_objects (see 6.26)
- load_simstate_behavior (see <u>6.27</u>)
- load_upf (see <u>6.28</u>)
- load_upf_protected (see <u>6.29</u>)
- set_design_top (see <u>6.38</u>)
- set_design_attributes (see <u>6.37</u>)
- set_port_attributes (see <u>6.46</u>)
- set_scope (see 6.52)
- set_simstate_behavior (see <u>6.53</u>)
- set_partial_on_translation (see <u>6.44</u>)
- **upf_version** (see <u>6.54</u>)

All other UPF commands have both an immediate and a deferred effect. For these commands, the immediate effect is to add the command syntax to an internal structure for further processing. The deferred effect varies with the command, but typically contributes to construction of a power intent model reflecting the specification. This model is then applied to the design as appropriate for the tool involved.

One exception is the **save_upf** command (see $\underline{6.36}$), for which the deferred effect is generation of a UPF file describing the power intent for a given scope. This generation occurs after the power intent model has been fully constructed, so the generated UPF file is complete.

NOTE—This algorithm defines a reference model for UPF command processing, to illustrate how the interdependencies between design data and the UPF specification, and among UPF commands themselves, can be satisfied. A given tool may use a different algorithm as long as the overall effect is the same as this algorithm would present.

8.2 Data requirements

In addition to the UPF file(s) involved, UPF processing requires access to the following data:

- Elaborated design hierarchy
- UPF attribute specifications in HDL (if any)
- Library cell definitions

These data need to be available when UPF processing begins.

8.3 Processing phases

The following describes the detailed sequence of operations to process a UPF description, extract the power intent it specifies, and apply the power intent to a design for use in a verification or implementation tool.

8.3.1 Phase 1—read and resolve UPF specification

In this phase, the UPF commands are parsed and further processed to create a normalized representation of the UPF specification. This involves the following operations:

- a) Read and execute each UPF command as it is read in
 - 1) Resolve references to the design relative to the current scope
 - 2) Execute create_logic_port (see <u>6.16</u>), create_logic_net (see <u>6.15</u>), and connect_logic_net (see <u>6.10</u>)
 - 3) Execute **find_objects** commands (see <u>6.26</u>) on elaborated design (unmodified)
 - 4) Build/extend syntactic model of UPF specification
- b) Augment syntactic model of UPF specification with HDL-specified and library-specified UPF attributes
- c) Collapse **-update** commands and check for conflicts
- d) Apply defaults for defaultable options

In general, names shall be defined before being referenced. In this phase, name-defining UPF commands are associated with the scope in which the object is defined, or with the parent object for which a subordinate object is defined, as appropriate, so that subsequent name references can be resolved at this stage.

Names of design objects referenced in UPF commands shall be defined in the design hierarchy before they are referenced in UPF. Names of the library cells referenced in UPF commands shall be defined for the design before they are referenced in UPF. Names of UPF-defined objects shall be defined and associated with the appropriate design hierarchy scope before they are referenced in UPF. Names of objects that are associated with other objects (supply set handles of power domains; functions of supply sets or supply set handles; port states of ports; power states of supply sets, power domains, or modules; simstates of power states) shall be defined and associated with the relevant parent object before they are referenced in UPF. Names of VCTs shall be defined in UPF and associated with the global VCT scope before they are referenced in UPF.

Any command that updates a previous command that defined a simple name in a design hierarchy scope shall be processed in the scope in which the original command was processed and be associated with that same scope. Any command that updates a previous command that defined an object associated with a parent object shall also be processed in the scope in which the original command was processed and be associated with that same parent object.

8.3.2 Phase 2—build power intent model

In this phase, the normalized UPF specification is executed to construct a model of the power intent expressed by the specification. This involves the following operations:

- a) Construct power domains
 - 1) As specified by **create_power_domain** commands (see 6.17)
 - 2) Using the effective element list algorithm in 5.10
 - 3) Including constructing required supply sets and functions
 - 4) Atomic power domains shall be constructed first, followed by non-atomic power domains
- b) Construct control logic for isolation, retention, and switch instances
 As specified by create_logic_* (see <u>6.15</u> and <u>6.16</u>) and connect_logic_net (see <u>6.10</u>) commands
- c) Construct supply networks and connections to power domains/strategies
 - 1) As specified by **create_supply_*** (see <u>6.20</u>, <u>6.21</u>, and <u>6.22</u>) and **create_power_switch** (see <u>6.18</u>) commands

- 2) connect_supply_* (see <u>6.11</u> and <u>6.12</u>), create_*_vct (see <u>6.14</u> and <u>6.23</u>), and associate_supply_set (see <u>6.7</u>) commands
 - i) Including equivalent supply declarations
 - ii) Including error checks related to supply set/function association
- d) Construct explicit, implicit, and automatic supply connections

As specified by **connect_supply_*** commands (see <u>6.11</u> and <u>6.12</u>), **associate_supply_set** (see <u>6.7</u>), etc.

- e) Apply the power model of a hard IP cell as specified by **apply_power_model** command (see <u>6.6</u>)
- f) Construct composite domains
 - 1) As specified by create_composite_domain (see <u>6.13</u>) commands
 - 2) Including propagation of primary supply to/among subdomains
 - 3) Including error checks related to domain composition
- g) Identify power-domain boundary ports and their supplies
 By analyzing the elaborated design and create_power_domain (see <u>6.17</u>) commands
- h) Apply retention strategies for each domain
 As specified by set_retention (see <u>6.49</u> and <u>4.5.6</u>)
- i) Apply repeater strategies for each domainAs specified by set_repeater (see <u>6.48</u> and <u>4.5.6</u>)
- j) Apply isolation strategies for each domain boundary portAs specified by set_isolation (see <u>6.41</u> and <u>4.5.6</u>)
- k) Apply level-shifting strategies for each domain boundary port As specified by set_level_shifter (see <u>6.43</u> and <u>4.5.6</u>)
- 1) Identify cells to use for isolation, level-shifting, retention, and switch elements As specified by **map** * (see 6.32 and 6.33) and **use interface cell** (see 6.55) commands
- m) Construct power statesAs specified by add_power_state (see <u>6.4</u>) commands
- n) Construct power state transitions
 As specified by describe_state_transition (see <u>6.24</u>) commands

8.3.3 Phase 3—recognize implemented power intent

In this phase, the **-instance** options of all commands are processed to identify instances of cells that implement the power intent. If a given command has a **-instance** option, this indicates that the command has been implemented by some preceding step in the flow. The implementation may or may not be complete. In particular, new logic added to the design by some tool step (e.g., for test insertion) may trigger further implementation through another application of the same command.

If a given command has a **-instance** option that specifies an empty string as the instance name, this indicates the instance resulting from applying the command in this particular context has been optimized away. In this case, tools shall not infer a cell for this application of the command. In particular, verification tools shall not infer a cell for purposes of verification, and implementation tools shall not re-implement the command by inserting a cell again.

If a given command has a **-instance** option that specifies a hierarchical name as the instance name, the specified instance shall exist in the design. It shall be an error if that hierarchical name does not identify a cell instance of the appropriate type for the command. Attributes specified in library cells, in HDL models, or in UPF may be used to determine whether a given cell instance is appropriate for the command whose

-instance option identifies it as resulting from the implementation of that command. In this case also, tools shall not infer a cell for this application of the command. Instead, the existing cell shall be used.

In addition to the preceding, commands that create supply or logic ports or nets are processed to identify any ports or nets that already exist in the HDL hierarchy. If a **create_supply_port** (see <u>6.21</u>), **create_supply_net** (see <u>6.20</u>), **create_logic_port** (see <u>6.16</u>), or **create_logic_net** (see <u>6.15</u>) command specifies a port or net name that already exists in the current scope of the HDL hierarchy, it shall be an error if that port or net name does not identify a port or net, respectively, of the appropriate type for the command. A supply port or net is appropriate for a **create_supply_port** or **create_supply_net** command, respectively, if it is declared to be of type <code>supply_net_type</code> defined in the package UPF. A logic port or net is appropriate if it is declared with the standard logic type in the relevant HDL. In this case also, tools shall not create a new port or net for this application of the command. Instead, the existing port or net shall be used.

8.3.4 Phase 4—apply power intent model to design

In this phase, some or all of the power intent model is applied to the HDL design. A given tool will add the power intent elements required for that tool's operation to the design model. Power intent model elements that are already present in the design will not be added again. This includes implementation of any checkers introduced by the **bind_checker** command (see <u>6.9</u>).

NOTE—It may be appropriate for a given tool to update existing elements in the design to more completely reflect the power intent model. For example, a tool may choose to change the data type of a net in the design used as a supply net, from a single-bit type to the appropriate (SystemVerilog or VHDL) supply_net_type.

8.4 Error checking

Error checking is done in various UPF processing stages. Error checks include the following classes of checks, which would be performed in Phases 1, 2, and 3 of UPF processing:

- a) Phase Read and resolve UPF specification (see 8.3.1)
 - 1) UPF syntax checks (including semantic restrictions)
 - 2) Update conflict checks
 - 3) Design scope/object reference checks (scope/object not found)
- b) Phase 2—Build power intent model (see <u>8.3.2</u>)
 - 1) Conflicts between two commands applying to same object
 - 2) Completeness checks (e.g., all instances are in a power domain)
- c) Phase 3—Identify implemented power intent (see 8.3.3)

Name conflicts (an existing design object conflicts with a UPF name)

If a tool detects and reports an error in any of the preceding UPF processing phases, the tool may continue processing if possible, in order to identify any additional errors that might exist in the UPF specification or its interpretation with the design hierarchy, but processing should terminate before Phase 4, where the power intent model is applied to the design hierarchy.

9. Simulation semantics

This clause details the simulation semantics for the UPF commands (see also <u>Clause 6</u>).

9.1 Supply network creation

UPF supply network creation commands define the power supply network that connects power supplies to the instances in a design. After these commands are applied, every instance in a design is connected to the power supply network. The *supply network* is a set of supply nets, supply ports, switches, and potentially, regulators and generators. Supply sets are defined in terms of supply nets and conveniently define a complete power circuit for instances. Supply sets simplify the management of related supply nets and facilitate connections based on the role the supply set provides for a power domain and the functions the supply nets provide within the set (see 9.2.2). The supply network defines how power sources are distributed to the instances and how that distribution is controlled.

A supply port that propagates but does not originate a supply state and voltage value defines a *supply source*. At any given time, a supply source can be traced through the supply network connectivity to a single root supply driver. The output port of a switch is a root supply source (with a corresponding driver); the value of its driver is computed according to the algorithm given in the following item <u>h</u>). HDL switch models should use the assign_supply2supply function to propagate the input supply to the output supply. assign_supply2supply propagates or maintains the trace back of the root supply driver information. Bias generators, voltage regulators, and switches modeled in HDL should create a root supply driver when the supply source originates from within the model.

Determination of the root supply driver is required for certain supply network resolution functions (see 6.20).

NOTE—Since the supply net type is defined in the package UPF, it is possible to create the supply network entirely in HDL source.

A supply net can be connected to a port declared in the HDL description. In this case, the supply net state is connected to the port; the voltage is not used. VCTs define the conversion from supply net state values to values of an HDL type and vice versa to facilitate more complex modeling consistent with an organization's logic value interpretations of UPF supply port states.

If a supply net is connected to a HDL port of a single bit type, a default VCT that maps the **FULL_ON** state to logic 1 and the **FULL_OFF** state to logic 0 shall be inserted automatically. The default VCT facilitates building simple functional models. If this mapping is not the one desired for a particular connection, a user-defined VCT implementing the desired mapping can be specified explicitly for the connection (see also <u>Annex F</u>).

Supply port/net interconnections create a supply network that may span multiple instances at potentially multiple levels in the logic hierarchy. Evaluation of supply networks during simulation requires consideration of the whole collection of electrically equivalent supply ports/nets (see <u>4.4.3</u>) making up each supply network.

- a) A group of electrically equivalent ports/nets (see <u>4.4.3</u>) constitutes a supply network, including ports/nets that are both equivalent by connection and declared electrically equivalent.
 - 1) The source(s) of the group are the top-level and leaf-level sources.
 - 2) The load(s) of the group are the top-level and leaf-level loads.
 - 3) Internal ports act only as connections within the group.
- b) If there are no resolved nets in the group, then the group is unresolved.
- c) For an unresolved group, it is an error if there is more than one supply source in the group.

IEEE Std 1801-2013

IEEE STANDARD FOR DESIGN AND VERIFICATION OF LOW-POWER INTEGRATED CIRCUITS

- d) If there is at least one resolved net in the group, then the group is resolved.
- e) For a resolved group, it is an error if
 - 1) the group contains two resolved nets with different resolution types;
 - 2) any two resolved nets in the group are separated by a unidirectional internal port.
- f) In general, it is an error if a unidirectional supply port (an input port or an output port) in the group
 - 1) has a supply source on the load side, and
 - 2) has a load on the supply source side.
- g) For an unresolved group of electrically equivalent supply ports/nets (see <u>4.4.3</u>), the single source drives all the loads directly.
- h) For a resolved group of electrically equivalent supply ports/nets
 - 1) all electrically equivalent resolved nets in a group are collapsed into a single resolved net;
 - 2) supply sources provide inputs to the resolved net;
 - 3) the resolution type of the resolved net determines how inputs are resolved;
 - 4) the resolved value is distributed to all loads.

9.2 Supply network simulation

9.2.1 Supply network initialization

Simulation initialization semantics are defined by each HDL. Existing models rely on the HDL initialization semantics for operations such as initializing ROMs, etc. To ensure that initialization of the design occurs correctly during power-aware simulation, model initialization code and design code should be cleanly separated. In Verilog or SystemVerilog, initial blocks can be used for model initialization code, since these are not affected by power-aware simulation semantics. In VHDL, model initialization code should be placed in processes that will not be synthesized and these processes should be included in an "always-on" power domain during power-aware simulation.

The initial state of supply ports and supply nets is **OFF** with an unspecified voltage value. The initial state of a supply set is determined by the initial state of each supply function of the supply set. The initial state of a supply set function is determined by the initial state of the corresponding supply net with which it has been associated or else the initial state of the root supply driver of that function.

NOTE—Implicitly created supply nets are initialized the same as explicitly created supply nets.

To facilitate modeling of non-inferable behavior in HDLs that can be used in both a UPF simulation and a traditional non-UPF simulation, the following are provided:

— Predefined constant of Boolean type: UPF_POWER_AWARE.

The value of this constant is **TRUE** in a UPF simulation, otherwise it is **FALSE**. This constant value is globally static only in a UPF simulation; i.e., its value is known at the time that SystemVerilog and VHDL generate statements are evaluated allowing the ability to specify logic that is conditionally generated only in a UPF simulation.

 In VHDL, a signal and, in SystemVerilog, a variable of type power_state_simstate can be declared within an architecture or module.

The name of this signal/variable shall be upf_simstate. upf_simstate can be used in a process's sensitivity list. It shall be an error if upf_simstate is assigned or connected to a port it can only be used locally and in a read-only context. In a UPF simulation, upf_simstate shall represent the active simstate of the supply set that is implicitly, automatically, or explicitly connected to the instance when simstate behavior has been enabled for that element. If simstate behavior is disabled for the element, then upf_simstate shall remain the constant value **CORRUPT**.

9.2.2 Power-switch evaluation

During simulation, a power switch created with create_power_switch corresponds to a process that is sensitive to changes in its input port (net state and voltage value), as well as its control ports. [A general introduction to power-switch behavior is described here (see 6.18 for the complete power-switch semantics).] Whenever the signals on the control ports change, the corresponding on-state Boolean functions are evaluated. If an on-state function evaluates *True*, the switch is closed, which causes the state of its input port to propagate to the output port (or for a multiplexed switch, the corresponding input is switched to the output), otherwise the switch is opened—the output supply port is assigned the state OFF and the voltage value is unspecified. If any of the control signals is X or Z, the input supply port is **UNDETERMINED**, the control signals match one of the error-state Boolean functions, or more than one on-state function evaluates *True*, then the behavior of the output supply port is assigned the state **UNDETERMINED**, the voltage level shall be unspecified, and the acknowledge ports shall be driven X; in this case, implementations may issue a warning or an error.

Example

Using the following create_power_switch command (see 6.18):

```
create_power_switch kb
-output_supply_port {outp pda_vdd}
-input_supply_port {inpl yt}
-input_supply_port {inp2 db}
-control_port {cp1 eh}
-control_port (cp2 as}
-on_state {yt_on_kb inp1 {(cp1 && !cp2)}}
-on_state {db_on_kb inp2 {(!cp1 && cp2)}}
-ack_port {ap yack 1}
```

creates an instance of an anonymous switch model that is functionally equivalent to the following SystemVerilog module definition:

```
import UPF::*;
module <anon> (
  output supply_net_type outp,
  output logic ap,
  input supply net type inpl, inp2,
input logic cp1, cp2 );
upf object handle in1H, in2H, outH;
initial begin
in1H = get object( "inp1" );
in2H = get object( "inp2" );
outH = get_object( "outp" );
if (!is valid handle( in1H ) || !is supply kind( in1H ) ||
   !is valid handle( in2H ) || !is supply kind( in2H ) ||
   !is valid handle( outH ) || !is supply kind( outH ))
 $display( "Invalid supply port connection on switch port");
end
always@(cp1, cp2, inp1, inp2)
  case ({cp1, cp2})
    01 : begin
           assign supply2supply( outp, inp2 );
           ap <= 1;
         end
```

```
10 : begin
           assign supply2supply( outp, inp1 );
           ap <= 1;
         end
    00 :
    11 :
         begin
           assign supply state( outp, OFF );
           ap <= 0;
         end
    default : begin
                assign supply state ( outp, UNDETERMINED );
                ap \leq X;
                $stop
              end
endmodule
```

The instance of the anon module is:

```
<anon> kb (.outp(pda_vdd), .inp1(yt), .inp2(db), .ap(yack), .cp1(eh),
.cp2(as));
```

9.2.3 Supply network evaluation

During simulation, each supply port and net maintains two pieces of information: a supply state and a voltage value. The supply state itself consists of two pieces of information: an on/off state and a full/partial state. The supply state values are **FULL_ON**, **OFF**, **PARTIAL_ON**, and **UNDETERMINED**. **PARTIAL_ON** typically represents a resolved supply net state when some, but not all, switches are **FULL_ON** or any switch is **PARTIAL_ON** (see also <u>6.20.2</u>).

During simulation, the supply network is evaluated repeatedly whenever the value of a root supply driver or a switch input changes. Supply network evaluation consists of the following:

- a) Evaluation and resolution of supply nets (see 6.20.2)
- b) Evaluation of power switches (see 6.18)
- c) Evaluation of supply set power states (see 9.3)
- d) Evaluation and application of simstates (see 9.4 and 9.5).

The supply network is evaluated in the same step of the simulation cycle as the logic network. New root supply driver values are propagated along the connected supply nets in the same manner that logic values are propagated along the logic network.

NOTE—As no material distinction between **PARTIAL_ON** and **PARTIAL_OFF** exists, only **PARTIAL_ON** is defined.

9.3 Power state simulation

9.3.1 Power state control

The power state of a root supply set may be changed from an HDL test bench in simulation using the set_power_state function defined in the package UPF (see <u>Annex B</u>). The set_power_state function changes the power state of the specified supply set (or supply set handle) to one of the states defined for the supply set (handle). This function can be used to control the supply states of root supply sets, before top-level supply networks have been implemented or completed.

When set_power_state is used to change a supply set's power state to a specified power state:

- a) It is an error if the specified power state is defined with either a logic expression or a supply expression.
- b) It is an error if any one of the supply set functions is associated with an explicitly declared supply net, either in the declaration of the supply set or via association of a supply set with a supply set handle.
- c) The implicitly created supply nets of the set (e.g., primary.power), shall have their state set as follows:
 - 1) If the simstate of the specified power state is **CORRUPT**: the state shall be set to **OFF** and the voltage value is unspecified.
 - 2) For any other simstate: the state shall be set to FULL_ON and the voltage value is unspecified.

The set_power_state function cannot be used to set the power state of a power domain. However, setting the power state of a supply set or supply set handle to a given power state may indirectly affect the power state of a power domain, just as would occur if the power state of the supply set or supply set handle changed to the given power state as a result of the state of the supply network driving the root supply sets.

NOTE—Tools may provide other mechanisms to change the power state of the supply set or power domain. Such mechanisms are outside the scope of this standard.

9.3.2 Power state determination

Each supply set and each power domain may have an associated set of named power states. Each named power state is defined in terms of the values of supply ports or nets, or the power states of other supply sets or power domains, or logic signals representing control conditions, or some combination thereof.

A supply set or power domain is in a given power state S at a given time T if the definition of S is satisfied at time T by the current values of any supply or logic ports or nets referenced in the definition and by the current power states of any supply sets or power domains referenced in the definition. More than one power state definition can be satisfied at the same time, so a supply set or power domain may be in multiple power states at any given time.

The power state of a supply set is determined after all signals (including supply nets; see 9.2.3) have been updated and prior to the evaluation of the power state(s) of power domains. The power state of a power domain is determined after the power state(s) of all supply sets have been determined and prior to evaluation of user-defined processes and always blocks.

The power state of a supply set (or supply set handle) is evaluated whenever there is

- a) a change in the value of any supply set (handle) function, supply net, or logic net referenced in any power state definition of the supply set, or
- b) a call to the set power state function for this supply set.

The power state of a supply set is determined as follows:

```
for a supply set SS
power state set CPS = {}
for each power state PS defined for SS
    if PS has neither a supply expression nor a logic expression, then
        if set_power_state was called to set the power state to PS, then
            CPS = CPS + {PS}
        end
    else if PS has a supply expression but no logic expression, then
        if the supply expression is True, then
            CPS = CPS + {PS}
```

```
end
   else if PS has a logic expression but no supply expression, then
        if the logic expression is True, then
           CPS = CPS + \{PS\}
        end
   else (PS has both a logic expression and a supply expression)
         if the logic expression is True, then
           CPS = CPS + \{PS\}
           if the supply expression is False, then
              Error: Supply status insufficient to support power state
           end
        end
     end
  end
  if CPS = \{\}, then
     CPS = CPS + {DEFAULT CORRUPT}
  end
  current power states of SS = CPS
end
```

The power state of a power domain is evaluated whenever there is

- c) a change in the set of current power states of any supply set (handle) or other power domain referenced in any power state definition of the power domain, or
- d) a change in the value of any supply net or logic net referenced in any power state definition of the power domain.

The power state of a power domain is determined as follows:

9.4 Simstate simulation

The current simstate of a supply set (or supply set handle) is reevaluated whenever there is a change in the set of current power states of the supply set. If no power state in the set defines a simstate, then the current simstate remains unchanged. Otherwise, the current simstate of the supply set is set to the most corrupting simstate defined for any power state in the set of current power states of the supply set.

Each simstate has well-defined simulation semantics, as specified in the following subclauses. Multiple power states may be defined with the same simstate specification. The simstate semantics are applied to all elements that have the supply set connected to it (including no supply net connections except those implied by the supply set connection to the element) and that have the simstate semantics implicitly or explicitly enabled.

Elements implicitly connected to a particular supply set have simstate semantics enabled by default. Elements automatically or explicitly connected to a particular supply set have simstate semantics disabled by default. Use **set_simstate_behavior** to override the default enablement of simstate semantics (see <u>6.53</u>).

The supply set powering a state element or the driver for a net may be in a state that the supply is not adequate to support normal operational behavior. Under specified circumstances while in these states, the logic value of the state element or net becomes unknown. A corrupt value for a state element or net indicates the logic state of the state element or net is unknown due to the state of the supply powering the state element or driver of the net. The corrupt value of a state element or net shall be the HDL's default initial value for that object's type, except for VHDL std_ulogic and std_logic typed-objects, which shall use X as the corruption value (not U).

NOTE—An object may be declared with an explicit initial value. This explicit initial value has no relationship to the corrupt value for the object. For example, in VHDL, the objects of Integer type have the default initial value of Integer'Left (-2147483648 for a system using 32 bits to represent Integer types). A process variable inferring a state element may be declared to be of type Integer with an initial value of 0. The corrupt value for the variable is Integer'Left, not 0.

The following subclauses define the simulation semantics for simstates. These semantics are applied to the elements connected to the supply set with simstate behavior **ENABLED**.

9.4.1 NORMAL

This state is a normal, power-on functional state. The simulator executes the design behavior of the elements consistent with the HDL or UPF specification that defines the element.

9.4.2 CORRUPT

This state is a non-functional state. For example, this state can be used to represent a power-gated/power-off supply set state. In this power state, state elements powered by the supply set and the logic nets driven by elements powered by the supply set are corrupted. The element is disabled from evaluation while this state applies.

As long as the supply set remains in a **CORRUPT** simstate, no additional activity shall take place within the elements, i.e., all processes modeling the behavior of the element become inactive, regardless of their original sensitivity list. Events that were scheduled for elements supplied by the supply set before entering this simstate shall have no effect.

9.4.3 CORRUPT_ON_ACTIVITY

This state is a power-on state that is not dynamically functional. For example, this state can be used to represent a high-voltage threshold, (body-bias) state that does not have characterized (defined) switching performance. In this simstate, the logic state of the elements is maintained unless there is activity on any of the element's inputs. Upon activity on any input, then all state elements and logic nets driven by the element are corrupted.

9.4.4 CORRUPT_ON_CHANGE

This state is a power-on state that is not dynamically functional. For example, this state can be used to represent a high-voltage threshold, (body-bias) state that does not have characterized (defined) switching performance. In this simstate, the logic state of the elements is maintained unless there is a change on any of the element's outputs. Upon change of any output, then all logic nets driven by that element output are corrupted.

9.4.5 CORRUPT_STATE_ON_CHANGE

This state is a power-on state that represents a power level sufficient to power normal functionality for combinational functionality, but insufficient for powering the normal operation of a state element if the state element is written with a new value. The simulator executes the design behavior of the elements consistent with the HDL or UPF specification that defines the element, except that any change to the stored value in a state element results in the writing of a corrupt value to the state element.

9.4.6 CORRUPT_STATE_ON_ACTIVITY

This state is a power-on state that represents a power level sufficient to power normal functionality for combinational functionality but insufficient for powering the normal operation of a state element if there is any write activity on the state element. The simulator executes the design behavior of the elements consistent with the HDL or UPF specification that defines the element, except that any activity inside state elements, whether that activity would result in any state change or not, results in the writing of a corrupt value to the state element.

9.4.7 NOT_NORMAL

This is a special, placeholder state. It allows early specification of a non-operational power state while deferring the detail of whether the supply set is in the CORRUPT, CORRUPT_ON_ACTIVITY, CORRUPT_ON_CHANGE, CORRUPT_STATE_ON_CHANGE, or CORRUPT_STATE_ON_ACTIVITY simstate. If the supply set matches a power state specified with simstate NOT_NORMAL, the semantics of CORRUPT shall be applied, unless overridden by a tool-specific option. NOT_NORMAL semantics shall never be interpreted as NORMAL.

The functions defined in package UPF (see <u>Annex B</u>) that query the simstate for a state that was originally **NOT_NORMAL** shall return the simstate to be applied in simulation for that state. e.g., **CORRUPT** for the default interpretation of **NOT_NORMAL**.

The query functions (see <u>Annex C</u>) that query the simstate for a state having a **NOT_NORMAL** simstate shall return **NOT NORMAL** when it was not updated with any other simstate.

NOTE 1—Using the default interpretation of **CORRUPT** for **NOT_NORMAL** provides a conservative—the broadest corruption semantics—for simulation of the design for functional verification. However, a conservative interpretation of **NOT_NORMAL** for other tools, such as power estimation tools, might be to use a bias or lowered voltage level interpretation such as **CORRUPT_ON_ACTIVITY**.

NOTE 2—As it is possible for two or more power states of a supply set to match the state of the supply set's nets and for multiple simstate specifications to apply simultaneously, the effective result is that the simstate with the broadest corruption semantics shall apply. For example, a supply set that matches power states with simstates of **CORRUPT_STATE_ON_CHANGE** and **CORRUPT_STATE_ON_ACTIVITY** shall result in the application of **CORRUPT_STATE_ON_ACTIVITY** simstate semantics being applied.

9.5 Transitioning from one simstate state to another

The following subclauses define the simulation semantics for transitions from one simstate to another. These semantics are applied to the elements connected to the supply set with simstate behavior **ENABLED**.

9.5.1 Any state transition to CORRUPT

In this case, the nets and state elements driven by the elements connected the supply set in this simstate shall be corrupted. The elements connected to this supply set are inactive as long as the supply set is in the **CORRUPT** simstate.

9.5.2 Any state transition to CORRUPT_ON_ACTIVITY

In this case, the current state of nets and state elements driven by the element shall remain unchanged at the transition. The processes modeling the behavior of the element shall remain enabled for activation (evaluation). Any net or state element that is actively driven after transitioning to this state shall be corrupted.

Any attempt to restore a retention register's retained value while in the **CORRUPT_ON_ACTIVITY** state shall result in corruption of the register's value.

9.5.3 Any state transition to CORRUPT_ON_CHANGE

In this case, the current state of nets and state elements driven by the element shall remain unchanged at the transition. The processes modeling the behavior of the element shall remain enabled for activation (evaluation).

9.5.4 Any state transition to CORRUPT_STATE_ON_CHANGE

In this case, the current state of nets and state elements driven by the element shall remain unchanged at the transition. The processes modeling the behavior of the element shall be enabled for activation (evaluation).

9.5.5 Any state transition to CORRUPT_STATE_ON_ACTIVITY

In this case, the current state of nets and state elements driven by the element shall remain unchanged at the transition. The processes modeling the behavior of the element shall be enabled for activation (evaluation).

9.5.6 Any state transition to NORMAL

In this case, the processes modeling the behavior of the element shall be enabled for activation (evaluation), and the combinational and level-sensitive sequential logic functionality in each process shall be re-evaluated to restore and properly propagate constant values and current input values. Edge-sensitive sequential logic functionality within the element shall not be evaluated at this transition.

9.5.7 Any state transition to NOT_NORMAL

NOT_NORMAL is simulated according to the interpretation of this placeholder simstate (see 9.4.7).

9.6 Simulation of retention

This subclause covers some of the basics of retention register operation and modeling, which are useful in describing the simulation semantics for the **set_retention** command (see 6.49). The following abbreviations are used in various figures and tables herein:

VDDprimary supply port of the registerVDDRETretention supply port of the registerSSsave signal is activeSCsave conditionRSrestore signal is active

- RC restore condition
- RTC retention condition

9.6.1 Retention corruption summary

A retention register has the same simulation behavior as a regular register when both supplies VDD and VDDRET are ON, the save/restore signals are inactive and the retention condition is *False*. The main simulation difference between a non-retention register and a retention register comes when the corruption behavior is modeled during various power state transitions. The retention register is composed of at least three components (see 4.3.4), as follows:

- *Register value* is the data held in the storage element of the register. In functional mode, this value gets updated on the rising/falling edge of clock or gets set or cleared by set/reset signals, respectively.
- *Retained value* is the data in the retention element of retention register. The retention element is powered by the retention supply.
- *Output value* is the value on the output of the register.

The retained value of the retention register can be corrupted in the following ways:

a) If VDDRET==OFF

Corrupt if RET SUP COR is set

- b) Else If VDDRET==ON
 - 1) If VDD==ON

(SS && SC) && (RS && RC) (both save/restore are true) and SAV RES COR is set

- 2) Else If VDD==OFF
 - i) (SS && SC) trying to save when domain off
 - ii) (RS && RC) trying to restore when domain off
 - iii) !RTC

The output value of the retention register can be corrupted in the following ways:

c) If -use_retention_as_primary is specified

Output is corrupted whenever retained value (described above) is corrupted.

d) If -use_retention_as_primary is not specified

1) If VDD==OFF

Corrupt always

2) Else If VDDRET==OFF

Corrupt if RET SUP COR is set

In summary, the preceding algorithm covers all the conditions by which a retention register (i.e., retained value/output value) can be corrupted. A corrupted retention register can then be restored to a valid state by a combination of one or more of the following:

- Restore (power up) the corrupting supplies
- Deassert save/restore signals if the corruption is due to the condition when both are true simultaneously
- Deassert retention condition
- Apply reset/set and/or clock

9.6.2 Retention modeling for different retention styles

Depending on the type of retention, the controlling inputs of the retention register like the save/restore signals may or may not exist on the register boundary. Thus, it is important to understand the modeling of the different flavors of retention, namely balloon-style retention and master/slave-alive style retention (see 4.3.4).

When the **set_retention** (see <u>6.49</u>) is specified with **-save_signal** and (or) **-restore_signal**, balloon-style retention semantics are applied to it. The process of saving/restoring is unique to balloon-style retention. When the **set_retention** is not specified with both **-save_signal** and **-restore_signal** and it is specified only with a **-retention_condition**, the master/slave-alive retention semantics are applied instead. In this type of retention, the restore happens during power-up, as the master/slave latch is kept on the retention supply. However, whether to be in a retention state or not may be controlled by the value of one or more ports on the retention register.

A retention register may be in one of the following states:

NORMAL—Functional/active mode, all supplies expected to be ON.

SAVE—The time snapshot where the save action occurs (for balloon-latch style registers).

RESTORE—The time snapshot where the restore action occurs (for balloon-latch style registers).

RETAIN_ON—The time snapshot where the primary supply is ON and the register is in retention state (retention_condition == True).

RETAIN_OFF—The time snapshot where the primary supply is OFF and the register is in retention state (retention_condition == True).

PARTIAL_CORRUPT—The retained value is corrupted, but the register value is not corrupted. **CORRUPT** - The register value and retained value are both corrupted.

<u>Table 7</u> summarizes the power state of a balloon style retention register with respect to the states of the signals.

<u>Table 8</u> summarizes the power state of a master/slave alive retention register with respect to the states of the signals.

Table 9 shows the output values of the retention register depending on the state of retention register.
Comments					SAV_RES_COR is		RET_SUP_COR is	RET_SUP_COR is	RET_SUP_COR is	RTC		Restore during bower-down	Save during power- lown	
Valid next states	SAVE, RESTORE	NORMAL, RETAIN OFF, RESTORE	NORMAL, RETAIN_ON	RETAIN_ON, NORMAL	NA	NA	NA I	NORMAL	NA 8	NA	RETAIN_ON	NA I	NA	
Register state	NORMAL	RETAIN_ON	RESTORE	SAVE	CORRUPT	CORRUPT	CORRUPT	PARTIAL CORRUPT	CORRUPT	CORRUPT	RETAIN_OFF	CORRUPT	CORRUPT	
Register value	Previous state value	Previous state value	Retention value	Previous state value	CORRUPT	CORRUPT	CORRUPT	Previous state value	CORRUPT	CORRUPT	CORRUPT	CORRUPT	CORRUPT	g next states.
Retained value	Previous saved data	Previous saved data	Previous saved data	Register value	CORRUPT	CORRUPT	CORRUPT	CORRUPT	CORRUPT	CORRUPT	Previous saved data	CORRUPT	CORRUPT	s are non-corruptin
RTC	FALSE	TRUE	×	×	×	TRUE	FALSE	FALSE	×	FALSE	TRUE	×	Х	alid next state
RS && RC	FALSE	FALSE	TRUE	FALSE	TRUE	X	TRUE	FALSE	×	FALSE	FALSE	TRUE	Х	² condition. V
SS && SC	FALSE	FALSE	FALSE	TRUE	TRUE	X	X	×	×	FALSE	FALSE	FALSE	TRUE	a "don't-care'
VDD RET	NO	NO	NO	NO	NO	OFF	ΟFF	OFF	OFF	NO	NO	NO	NO	table denotes
QQV	NO	NO	NO	NO	NO	NO	NO	NO	OFF	OFF	OFF	OFF	OFF	^a The X in this

retention ^a
style
balloon
for
table
state
power
7—Retention
Table

VDD	VDD RET	RTC	Retained/ register value	Register state	Valid next states	Comments
ON	ON	FALSE	Previous state value	NORMAL	RETAIN_ON	_
ON	ON	TRUE	Previous state value	RETAIN_ON	NORMAL, RETAIN_OFF	—
ON	OFF	TRUE	CORRUPT	CORRUPT	NA	RET_SUP_COR is set
ON	OFF	FALSE	CORRUPT	CORRUPT	NA	RET_SUP_COR is set
OFF	OFF	Х	CORRUPT	CORRUPT	NA	
OFF	ON	FALSE	CORRUPT	CORRUPT	NA	!RTC
OFF	ON	TRUE	Retention value	RETAIN_OFF	RETAIN_ON	

Table 8—Retention state table for master/slave-alive retention

Table 9—Retention output value table^a

use_retention_ as_primary	State	Register value	Output value
TRUE	NORMAL	DATA	DATA
TRUE	RETAIN-ON/RETAIN-OFF	DATA	DATA
TRUE	SAVE	DATA	DATA
TRUE	RESTORE	DATA	DATA
TRUE	CORRUPT	Х	Х
FALSE	NORMAL	DATA	DATA
FALSE	RETAIN-ON/RETAIN-OFF	DATA	VDD==ON?DATA:X
FALSE	SAVE	DATA	VDD==ON?DATA:X
FALSE	RESTORE	DATA	VDD==ON?DATA:X
FALSE	CORRUPT	Х	Х

^aDATA in <u>Table 9</u> stands for a valid data, and X stands for corrupt data.

Figure 6 describes the sequence of transitions in balloon style retention register. In this case, the state transitions are not synchronous, i.e., they are not caused due by clock transitions.

Figure 7 describes the sequence of transitions in a master/slave-alive register. In this case, the state transitions are not synchronous, i.e., they are not caused due by clock transitions.



Figure 6—Retention state transition diagram for balloon-style retention



Figure 7—Retention state transition diagram for master/slave-alive style retention

9.7 Simulation of isolation

The simulation semantics for isolation are defined through an equivalent SystemVerilog always block, unless **-instance** applies to a specific isolation element or **use_interface_cell** (see <u>6.55</u>) is applied.

An isolation strategy with a constant clamp value (0, 1, Z, or a user-specified value) is functionally equivalent to the following SystemVerilog code:

The isolation cell with a clamp value of latch is functionally equivalent to the following SystemVerilog code:

9.8 Simulation of level-shifting

A level-shifter has the logical functionality of a buffer.

9.9 Simulation of repeater

A repeater has the logical functionality of a buffer.

Annex A

(informative)

Bibliography

Bibliographical references are resources that provide additional or helpful material but do not need to be understood or used to implement this standard. Reference to these resources is made for informational use only.

[B1] IEEE Standards Dictionary Online.⁸

[B2] IEEE Std 1364[™], IEEE Standard for Verilog Hardware Description Language.^{9, 10}

[B3] IEEE Std 1801[™]-2009, IEEE Standard for Design and Verification of Low Power Integrated Circuits.

[B4] ISO/IEC 8859-1, Information technology—8-bit single-byte coded graphic character sets—Part 1: Latin Alphabet No. 1.¹¹

[B5] Tcl language syntax summary.¹²

[B6] Tcl language usage.¹³

[B7] Liberty library format usage.¹⁴

⁸*IEEE Standards Dictionary Online* subscription is available at:

 $[\]underline{http://www.ieee.org/portal/innovate/products/standard/standards_dictionary.html.}$

⁹IEEE publications are available from The Institute of Electrical and Electronics Engineers (<u>http://standards.ieee.org/</u>).

¹⁰The IEEE standards or products referred to in this clause are trademarks of the Institute of Electrical and Electronics Engineers, Inc.

¹¹ISO/IEC publications are available from the ISO Central Secretariat (http://www.iso.org/). ISO publications are also available in the United States from the American National Standards Institute (http://www.ansi.org/).

¹²Available at http://www.tcl.tk/man/tcl8.4/TclCmd.

¹³Available at http://sourceforge.net/projects/tcl/.

¹⁴Available at http://opensourceliberty.org/opensourceliberty.html.

Annex B

(normative)

HDL package UPF

B.1 Supply net logic type values

These functions are required for any implementations supporting VHDL and/or SystemVerilog simulation.

The real typed value parameter to the **supply_on** and **supply_partial_on** functions is the voltage value in units of volts. This voltage value shall be converted into a signed integer value in units of microvolts.

B.2 Path names

Any string representing a hierarchical path name need to use a slash (/) as the hierarchy delimiter.

B.3 VHDL UPF package

The following defines the VHDL package for UPF. This package shall be located in the IEEE library:

```
Library IEEE;
Use IEEE.std logic 1164.all;
Use IEEE.numeric bit.all;
package UPF is
  type state is (OFF,
                     UNDETERMINED,
                     PARTIAL ON,
                     FULL ON);
  -- The provided routines shall be used to ensure
  -- the HDL code is independent of the details of the supply net
  -- type implementation. This ensures portability and forward
  -- compatibility of the HDL.
  -- The supply net type implementation is openly specified for
  -- the following reasons:
  ___
     1. Users know how supply net and port values will visually
          appear in tools such as wave windows.
  ___
     2. C language access by user or 3rd party tools can depend
  ___
  ___
          on existing functionality to read and write supply
  ___
          values.
  ___
  -- Tools implementing this package may optimize the supply data
  -- type as long as the 2 items above are preserved and the
  -- supply value set and get routines are supported.
  type supply net type is record
   state : state;
    -- Voltage in microvolts
   voltage : signed(31 downto 0);
  end record;
```

```
-- Types used to navigate and to find UPF objects in
-- the design hierarchy
subtype upf object handle is Integer;
type object kind is (ERROR KIND,
                     SWITCH, ISOLATION CELL, LEVEL SHIFTER,
                     SUPPLY SET, SUPPLY NET, SUPPLY PORT,
                     ROOT SUPPLY DRIVER,
                     LOGIC NET, LOGIC PORT,
                     INSTANCE,
                     POWER DOMAIN,
                     UPF POWER STATE,
                     ITERATOR,
                     OTHER );
-- NOTE: UNDETERMINED is not defined as a power state kind as
-- it is replaced during simulation with a determined state
type power state kind is
  (ERROR PS, OPERATING, ILLEGAL, TRANSIENT);
type power state simstate is
 (NORMAL, CORRUPT, CORRUPT_ON_ACTIVITY, CORRUPT_ON_CHANGE,
 CORRUPT STATE ON CHANGE, CORRUPT STATE ON ACTIVITY);
subtype supply kind is object kind
 range SUPPLY NET to ROOT SUPPLY DRIVER;
-- Voltage is a real value in volts that is converted into
-- an integer value normalized to microvolts
function supply on (
                          -- Path name to supply net, port or
 supply name : STRING;
                          -- root supply driver
           : REAL := 1.0 )
 voltage
return BOOLEAN;
function supply off (
 supply name : STRING )
return BOOLEAN;
-- Voltage is a real value in volts that is converted into
-- an integer value normalized to microvolts
function supply_partial_on (
 supply name : STRING;
 value
        : REAL := 1.0 )
return BOOLEAN;
function get supply value (
 supply name : STRING )
return supply_net_type;
function get supply voltage (
 value : supply_net_type )
return REAL;
function get_supply_on_state (
 value : supply_net_type )
return BOOLEAN;
```

```
function get supply on state (
 value : supply net type )
return BIT;
function get supply state (
 value : supply net type )
return state;
-- Routines to navigate and find UPF objects in the design hierarchy
-- The initial scope shall be the root of the simulation
-- which allows access to the testbench as well as design
-- under verification.
-- If inst path is valid for the current scope, then
-- the function changes the scope to that instance.
-- The function returns TRUE on success, FALSE if the
-- the scope cannot be set as requested.
function set scope( inst path : STRING )
return Boolean;
-- This function returns the current scope's complete
-- instance path from the root of the simulation.
function get scope
return STRING;
-- Tests the handle and returns TRUE if the handle is valid
-- and FALSE if it is invalid
function is valid handle ( handle : in upf object handle )
return Boolean;
-- Get a handle to a design object (either HDL or UPF created).
-- Returns a valid handle on success; invalid handle on failure
function get object( inst path : STRING;
return upf_object_handle;
-- Returns the kind of object that the handle refers
-- to.
-- If the handle is not valid, ERROR object kind is
-- returned.
function get_object_kind( handle : upf_object_handle )
return object kind;
-- Returns TRUE if the kind of object referenced by
-- handle is a supply_net, supply_port or
-- root supply driver.
-- Returns FALSE otherwise.
function is supply kind ( handle : upf object handle )
return Boolean;
-- For a supply kind of object referenced by handle,
-- return the state of that object.
-- It is the caller's responsibility to ensure that
-- the handle passed references a supply kind of object.
-- If the object is not a supply kind, the value returned
-- is UNDETERMINED
function get_supply_state( handle : upf_object_handle )
return state;
```

IEEE Std 1801-2013

IEEE STANDARD FOR DESIGN AND VERIFICATION OF LOW-POWER INTEGRATED CIRCUITS

-- For a supply kind of object referenced by handle, -- return the voltage of that object. -- It is the caller's responsibility to ensure that -- the handle passed references a supply kind of object. -- If the object is not a supply kind, the value returned -- is -1.0. function get_supply_voltage(handle : upf_object handle) return REAL; -- For a handle that references a supply kind object, sets -- the net state and voltage of the supply. -- Returns TRUE on success. -- Returns FALSE if the supply state cannot be set or -- if the object that handle references is not a supply -- kind of object. function assign_supply_state(handle : upf_object_handle; state := OFF; voltage : REAL := 0.0, delay : TIME := 0 ns) return Boolean; -- Quick checks for the information specified by the -- function name. -- All functions return TRUE if the information/state -- specified is true for the object referenced by handle. -- Returns FALSE if it is not true or if the information/ -- state being compared or check is not applicable to the -- kind of object that handle references. function is supply full on (handle : upf object handle) return Boolean; function is supply off (handle : upf object handle) return Boolean; function is supply partial on (handle : upf object handle) return Boolean; function is supply undetermined (handle : upf object handle) return Boolean; function is_supply_equal (handle : upf_object_handle; state : state; voltage : real) return Boolean; -- Both handles shall reference a supply kind of object. -- Returns TRUE if states are the same and, if -- state is not OFF, the voltages are the same. -- This function does not check the root supply drivers of -- the supplies or any other connectivity aspects of the supplies function are supplies equivalent (handle1 : upf object handle; handle2 : upf object handle) return Boolean; -- Assigns the source supply to the destination supply. -- For purposes of supply net resolution, the destination -- will be sourced by the same root supply driver as the source. -- (The source may be a root supply driver.) -- Returns TRUE on success, FALSE on failure.

```
function assign supply2supply( destination : upf object handle;
                               source : upf object handle;
                               delay : TIME := 0 ns )
return Boolean;
-- Creates a root supply driver than can be used to drive
-- one or more supply nets from within an HDL model of a supply
-- network component (HDL model of a bias generator, for example).
-- The root supply driver is created within the scope of the parent.
-- The parent and driver name information may be used for error reporting.
-- Returns a valid object handle on success and an invalid object handle
-- on failure.
function create root supply driver (
 driver name : STRING;
 parent : upf object handle )
return upf object handle;
-- Routines to query and set power states on various objects.
-- There can be 0, 1 or many power states defined for a given
-- object. The iterator provides a mechanism to retrieve a
-- an opaque list handled by the tool.
-- If there are 0 power states, then the handle returned is
-- an invalid handle.
function get iterator for all ps ( handle : upf object handle )
return upf object handle;
-- Returns an iterator referencing all power states of the
-- specified object that are active when the call is made.
-- The returned handle is invalid if there are no power states
-- defined for the specified handle or if none of the power
-- states defined are active.
function get_iterator_for_all_active_ps (
 handle : upf_object_handle )
return upf object handle;
-- If there are more items in the iterator, this routine
-- will return the next item in the iterator.
-- Otherwise, an invalid handle will be returned if there
-- are no more objects to iterate over or if the iterator is
-- invalid
function iterate( iterator : upf object handle )
return upf object handle;
-- Returns the name of a power state kind of object.
-- Returns the null string if the handle does not reference
-- a power state object.
function get ps name ( power state : upf object handle )
return STRING;
-- For a handle referencing a power state object,
-- return the kind of power state.
-- Returns ERROR if the handle is invalid or does
-- not reference a power state object
function get ps kind( power state : upf object handle )
return power state kind;
-- For a handle referencing a power state object,
-- return the simulation state associated with the power state.
```

```
function get ps simstate ( power state : upf object handle )
 return power state simstate;
 -- Returns TRUE if the object to which this power state is
 -- attributed is in a state consistent with being in this
 -- power state.
 -- Returns FALSE otherwise (including if the power_state
 -- handle is invalid)
 function is active (power state : upf object handle )
 return Boolean;
 -- Returns TRUE if the object referenced by handle
 -- is in the power state referenced by the power state
 -- handle. If either handle is invalid, it returns FALSE.
 function is_in ( handle
                           : upf_object_handle;
                  power_state : upf_object_handle )
 return Boolean;
 -- Set the object to the specified power state.
 -- This function returns TRUE on success.
 -- It returns FALSE on failure.
 -- The function will fail if
 -- a. the object is not a root supply set or supply set handle, or
 -- b. any function of the supply set is associated with an
 -- explicitly declared net.
 function set power state( object
                                     : upf object handle;
                            power state : upf object handle;
                            delay : TIME := 0 ns )
 return Boolean;
 -- Routines to facilitate type conversion of a supply net state to a
 -- logic value; specifically, for use in connecting a supply net to a
 -- logic port that is tied high or tied low.
 -- Returns 1 if the supply net is ON at any voltage level > 0.0.
 -- Returns X if the supply net is OFF or PARTIAL ON.
 -- It is up to the user to ensure that a proper supply net is
 -- connected to a power net.
 function tie hi ( supply net : supply net type )
 return std logic;
 -- Returns 0 if the supply net is OFF.
 -- Returns X if the supply net is ON or PARTIAL ON.
 -- It is up to the user to ensure that a proper supply net is
 -- connected to a ground net.
 function tie_lo ( supply_net : supply_net_type )
 return std logic;
end package UPF;
```

B.4 SystemVerilog UPF package

The following defines the SystemVerilog package for UPF:

package UPF;
// Bit encoding of the state type is provided
// for backward compatibility to UPF 1.0.

```
typedef enum {OFF = 0,
              UNDETERMINED,
              PARTIAL ON,
              FULL ON} state;
// The provided routines shall be used to ensure
// the HDL code is independent of the details of the supply net
// type implementation. This ensures portability and forward
// compatibility of the HDL.
// The supply net type implementation is openly specified for
// the following reasons:
11
    1. Users know how supply net and port values will visually
11
       appear in tools such as wave windows.
    2. C language access by user or 3rd party tools can depend
11
11
        on existing functionality to read and write supply
11
        values.
11
// Tools implementing this package may optimize the supply data
// type as long as the 2 items above are preserved and the
// supply value set and get routines are supported.
typedef struct packed {
 state state;
 int
          voltage; // voltage in microVolts
} supply_net_type;
// Types used to navigate and to find UPF objects in
// the design hierarchy
typedef chandle upf object handle;
typedef enum {ERROR KIND,
              SWITCH, ISOLATION CELL, LEVEL SHIFTER,
              SUPPLY SET, SUPPLY NET, SUPPLY PORT,
              ROOT SUPPLY DRIVER,
              LOGIC NET, LOGIC PORT,
              INSTANCE,
              POWER DOMAIN,
              UPF POWER STATE,
              ITERATOR,
              OTHER } object_kind;
// NOTE: UNDETERMINED is not defined as a power state kind as
         it is replaced during simulation with a determined state
11
typedef enum
 {ERROR PS, OPERATING, ILLEGAL, TRANSIENT} power state kind;
typedef enum
 {NORMAL, CORRUPT, CORRUPT ON ACTIVITY, CORRUPT ON CHANGE,
 CORRUPT STATE ON CHANGE, CORRUPT STATE ON ACTIVITY} power state simstate;
// SystemVerilog does not support subtype definitions
// Therefore, there is no equivalent to the VHDL subtype
// definition of supply kind.
// Voltage is a real value in volts that is converted into
// an integer value normalized to microvolts
// SystemVerilog does not support function overloading by
// input parameter type. Therefore, a 2nd version of functions
// is specified.
```

```
function bit supply on ( string pad name, real value = 1.0);
endfunction
function bit supply on from handle (
  upf object handle supply, real value = 1.0);
endfunction
function bit supply off( string pad name );
endfunction
function bit supply partial on( string pad name, real value = 1.0 );
endfunction
function supply_net_type get_supply_value( string name );
endfunction
function supply net type get supply value from handle (
  upf object handle supply );
endfunction
function real get supply voltage (supply net type arg );
endfunction
function bit get supply on state( supply net type arg );
endfunction
function state get supply state( supply net type arg );
endfunction
// Routines to navigate and find UPF objects in the design
// hierarchy
// The initial scope shall be the root of the simulation
\ensuremath{//} which allows access to the testbench as well as design
// under verification.
// If inst path is valid for the current scope, then
// the function changes the scope to that instance.
// The function returns TRUE on success, FALSE if the
// the scope cannot be set as requested.
function bit set scope( string inst path );
endfunction
// This function returns the current scope's complete
// instance path from the root of the simulation.
function string get scope( );
endfunction
// Tests the handle and returns TRUE if the handle is valid
// and FALSE if it is invalid
function bit is valid handle ( upf object handle handle );
endfunction
// Get a handle to a design object (either HDL or UPF created).
// Returns a valid handle on success; invalid handle on failure
function upf object handle get object(
  string inst_path);
endfunction
// Returns the kind of object that the handle refers to.
```

// If the handle is not valid, ERROR object kind is returned.

```
function object kind get object kind( upf object handle handle );
endfunction
// Returns TRUE if the kind of object referenced by
// handle is a supply net, supply port or
// root supply driver.
// Returns FALSE otherwise.
function bit is supply kind ( upf object handle handle );
endfunction
// For a supply kind of object referenced by handle,
// return the state of that object.
// It is the caller's responsibility to ensure that
// the handle passed references a supply kind of object.
// If the object is not a supply kind, the value returned
// is UNDETERMINED
function state get_supply_state_from_handle(
 upf object handle handle );
endfunction
// For a supply kind of object referenced by handle,
// return the voltage of that object.
// It is the caller's responsibility to ensure that
// the handle passed references a supply kind of object.
// If the object is not a supply kind, the value returned
// is -1.0.
function real get supply voltage from handle ( upf object handle handle );
endfunction
// For a handle that references a supply kind object, sets
// the net state and voltage of the supply.
// Returns TRUE on success.
// Returns FALSE if the supply state cannot be set or
// if the object that handle references is not a supply
// kind of object.
function bit assign supply state(
 upf object handle handle,
 state state = OFF,
 real voltage = 0.0,
 time delay := Ons );
endfunction
// Quick checks for the information specified by the function name.
// All functions return TRUE if the information/state
// specified is true for the object referenced by handle.
// Returns FALSE if it is not true or if the information/
// state being compared or check is not applicable to the
// kind of object that handle references.
function bit is supply full on ( upf object handle handle );
endfunction
function bit is supply off ( upf object handle handle );
endfunction
function bit is supply partial on ( upf object handle handle );
endfunction
function bit is_supply_undetermined ( upf_object_handle handle );
endfunction
```

```
function bit is supply equal (
 upf object handle handle,
 state
           state,
 real
                   voltage );
endfunction
// Both handles shall reference a supply kind of object.
// Returns TRUE if states are the same and, if
// state is not OFF, the voltages are the same.
// This function does not check the root supply drivers of
// the supplies or any other connectivity aspects of the supplies
function bit are supplies equivalent (
 upf object handle handle1,
 upf object handle handle2 );
endfunction
// Assigns the source supply to the destination supply.
// For purposes of supply net resolution, the destination
// will be sourced by the same root supply driver as the source.
// (The source may be a root supply driver.)
// Returns TRUE on success, FALSE on failure.
function bit assign supply2supply(
 upf object handle destination,
 upf object handle source,
                   delay := Ons );
 time
endfunction
// Creates a root supply driver than can be used to drive
// one or more supply nets from within an HDL model of a supply
// network component (HDL model of a bias generator, for example).
// The root supply driver is created within the scope of the parent.
// The parent and driver name information may be used for error reporting.
// Returns a valid object handle on success and an invalid object
// handle on failure.
function upf object handle create root supply driver (
 string
                   driver name,
 upf object handle parent );
endfunction
// Routines to query and set power states on various objects.
// There can be 0, 1 or many power states defined for a given
// object. The iterator provides a mechanism to retrieve a
// an opaque list handled by the tool.
// If there are 0 power states, then the handle returned is
// an invalid handle.
function upf object handle get iterator for all ps (
 upf object handle handle );
endfunction
// Returns an iterator referencing all power states of the
// specified object that are active when the call is made.
\ensuremath{//} The returned handle is invalid if there are no power states
// defined for the specified handle or if none of the power
// states defined are active.
function upf_object_handle get_iterator_for_all_active_ps (
 upf object handle handle );
endfunction
```

```
// If there are more items in the iterator, this routine
// will return the next item in the iterator.
// Otherwise, an invalid handle will be returned if there
// are no more objects to iterate over or if the iterator is
// invalid
function upf object handle iterate( upf object handle iterator );
endfunction
// Returns the name of a power state kind of object.
// Returns the null string if the handle does not reference
// a power state object.
function string get ps name( upf object handle power state );
endfunction
// For a handle referencing a power state object,
// return the kind of power state.
// Returns ERROR if the handle is invalid or does
// not reference a power state object
function power state kind get ps kind (
  upf object handle power state );
endfunction
// For a handle referencing a power state object,
// return the simulation state associated with the power state.
function power state simstate get ps simstate(
  upf object handle power state );
endfunction
// Returns TRUE if the object to which this power state is
// attributed is in a state consistent with being in this
// power state.
// Returns FALSE otherwise (including if the power state
// handle is invalid)
function bit is active ( upf object handle power state );
endfunction
// Returns TRUE if the object referenced by handle
// is in the power state referenced by the power state
// handle. If either handle is invalid, it returns FALSE.
function bit is in (
  upf object handle handle,
  upf object handle power state );
endfunction
// Set the object to the specified power state.
// This function returns TRUE on success.
// It returns FALSE on failure.
-- The function will fail if
-- a. the object is not a root supply set or supply set handle, or
-- b. any function of the supply set is associated with an
-- explicitly declared net.
function bit set power state (
  upf_object_handle object,
  upf_object_handle power_state,
  time
                    delay = Ons );
endfunction
```

// Routines to facilitate type conversion of a supply net state to a
// logic value; specifically, for use in connecting a supply net to a
// logic port that is tied high or tied low.
// Returns 1 if the supply net is ON at any voltage level > 0.0.
// Returns X if the supply net is OFF or PARTIAL_ON.
// It is up to the user to ensure that a proper supply net is
// connected to a power net.
function logic tie_hi (supply_net_type supply_net);
endfunction
// Returns 0 if the supply net is OFF.
// Returns 0 if the supply net is OFF.

// Returns X if the supply net is ON or PARTIAL_ON. // It is up to the user to ensure that a proper supply net is // connected to a ground net. function logic tie_lo (supply_net_type supply_net); endfunction

endpackage : UPF

Annex C

(normative)

Queries

This annex documents the syntax for each of the **query_*** commands. Each return value is a Tcl string object that is a list of defined objects, all options of the object, or individual settings for the object. The names returned (*Return values*) are relative to the current scope. If there are any names to be returned that are not rooted in the current scope, the query shall raise an "out-of-scope" error. This could occur, for example, if the power domain of an object was queried, but the scope of the domain that was to be returned via this query was not visible in the scope as specified by the active **set scope** command (see <u>6.52</u>).

- Each query in this clause consists of a keyword followed by one or more parameters. All parameters begin with a hyphen (-). The meta-syntax for the description of the syntax rules uses the conventions shown in <u>Table 1</u>.
- For general information on how errors are handled, see <u>5.12</u>.
- Since the queries only return information about the active design, they have no implementation or simulation semantics.
- Queries are not guaranteed to, and in virtually all situations do not, return information in the order that their corresponding command (see <u>Clause 6</u>) supplied it.
- Additional information can be returned by the queries, for example if an instance is added to a domain using add_domain_elements, then query_power_domain also returns this added element. Command refinement reconciliation is incorporated in query return values (see 5.11).
- All query_* commands search from the current scope down, unless otherwise stated.

query_upf and all **query_*** commands that accept the **-non_leaf** and **-leaf_only** options can be interpreted differently between tools depending upon the library source. For example, a simulation tool may have a hierarchical model representation of a IP block that is not returned if **-leaf_only** is specified (the search would traverse through this boundary to find leaf cells). However, an implementation tool could have this IP block represented as a timing abstract and thus could be treated as a leaf cell.

Query commands that have the **-detailed** option provide the ability to return information as a list of {*key value*} pairs. The *key* is derived from the argument name of the corresponding command (see <u>Clause 6</u>) that is being queried.

Commands that have a Boolean option, such as **-include_scope**, shall have a Boolean return flag of 1 if the option was specified and 0 if it was not. For commands that have arguments that accept Tcl lists, the query returns the entire list, e.g., -ports *list* produces the **-detailed** output of the form {ports {port_list_index_0} {port_list_index_1}{...}}. For commands that have arguments that have arguments that have lists containing optional arguments, e.g., -supply {*supply_set_handle* [*supply_set_ref*]} the query returns the optional argument (*supply_set_ref*) or a null string if the optional argument has not been specified, e.g., {supply_{set_ref}}...}.

When the **-detailed** argument to a query returns an argument for which no value has been specified, then the default value is returned. If there is no default, then a null list ({}) is returned.

NOTE—These **query_*** commands do not make up the *power intent* of a design; they are only used for *querying* the design database and are included in this standard to enable portable, user-specified query procedures across tools that are compliant to this standard.

C.1 query_upf

Purpose	Find objects (including U	PF created or inferred objects) in the logic hierarchy		
Syntax	<pre>query_upf <domain_name scope="" =""> -pattern search_pattern [-object_type <inst net="" port="" supply_net="" supply_port="" supply_set="" ="">] [-inst_type <level_shifter all="" isolation_cell="" retention_cell="" switch_cell="" ="">] [-direction <in inout="" out="" ="">] [-transitive [<true false="" ="">]] [-regexp -exact] [-ignore_case] [-non_leaf -leaf_only]</true></in></level_shifter></inst></domain_name></pre>			
	domain_name scope	Either a power domain or a scope can be specified. If a power domain is specified, the search is restricted to that power domain; otherwise. the search is restricted to the specified scope.		
	-pattern search_pattern	The string used for searching. By default, <i>search_pattern</i> is treated as a Tcl glob expression.		
	-object_type <inst <br="">port supply_port net supply_net supply_set></inst>	Limits the objects returned. By default, all objects are returned.		
Argumonts	-inst_type <level_shifter <br="">isolation_cell switch_cell retention_cell all></level_shifter>	If -object is inst , this option limits the type of instances returned to be level-shifter, isolation, switch, or retention cells. The default is all , which returns all instances.		
Arguments	-direction <in out="" ="" <br="">inout></in>	If -object is port , then -direction can be used to restrict the directions of the returned ports.		
	-transitive [<true <br="">FALSE>]</true>	If -transitive is not specified at all, the default is -transitive FALSE . If -transitive is specified without a value, the default value is TRUE .		
	-regexp -exact	-regexp enables support for regular expression in the specified <i>search_pattern</i> . -exact disallows wildcard expansion on the specified <i>search_pattern</i> . If neither -regexp or -exact are specified, then <i>search_pattern</i> is interpreted as a Tcl glob expression.		
	-ignore_case	Performs case-insensitive searches. By default, all matches are case sensitive.		
	-non_leaf -leaf_only	If -non_leaf is specified, only non-leaf instances are returned; if -leaf_only is specified, only leaf-level instances are returned. By default, both leaf and non-leaf instances are returned.		
Return value	Returns a list of names (re nothing is found that mate object names, without any of object.	elative to the current scope) of objects that match the search criteria; when ches the search criteria, a <i>null string</i> is returned. The list contains just the y indication of object type. The list may contain names of more than one type		

The **query_upf** command searches for instances, nets, supply nets, ports, and supply ports in and below the *scope* or within the extent of a *domain_name*. This command works on the logic hierarchy and can be executed post-UPF annotation.

The **query_upf** command works on the logic hierarchy from a domain-centric or hierarchy-centric approach. A *domain-centric approach* restricts the search to instances, net, or ports that are logically within the extent of the specified *domain_name*. A *hierarchy-centric approach* searches in the *scope* only, or in and below the *scope* when **-transitive** is specified.

A domain-centric search examines all logical levels that are members of the specified domain. Based on Figure C.1 and Figure C.2, the command query_upf {PD1} -pattern * looks for any object (port, net, or instance) matching the specified string in the logical hierarchies A, A/B, A/C, or A/B/D/F.



Figure C.1—Logic hierarchy



Figure C.2—Physical layout

If searching for inputs into PD3, the command

query upf {PD3} -pattern * -object type port -direction in

returns any inputs from $\{B \rightarrow D, F \rightarrow D, and E \rightarrow D\}$.

-inst_type only returns instances of a particular type. For example, to find all level-shifters in the domain PD3, the following **query_upf** command could be used:

query_upf {PD3} -pattern * -inst_type level_shifter -object inst

A domain-centric search examines all logical levels that are members of the *domain_name*. Based on Figure C.1 and Figure C.2, the command query_upf {PD1} -pattern *BW1* looks for any object (port, supply port, net, supply net, or instance) that matched the specified string in the logical hierarchies A, A/B, A/C, or A/B/D/F.

If searching for inputs into PD3, the command

query_upf {PD3} -pattern * -object port -direction in

returns any inputs from $\{B \rightarrow D, F \rightarrow D, and E \rightarrow D\}$.

The following conditions also apply:

- **-transitive** is ignored in a domain-centric search.
- The specified *domain_name* or *scope* cannot start with . . or /, i.e., **query_upf** shall be referenced from the current scope, and reside in the current scope or below it.
- All elements returned are referenced to the current scope.
- If *domain name* or *scope* is specified as . (a dot), the current scope is used as the root of the search.
- query_upf takes a *scope* argument. The specified scope may reference a generate block as the root of the search.
- For details on pattern matching and wildcarding, see <u>6.26.1</u> and <u>Table 5</u>.

Syntax examples:

```
query_upf A/B/D \
-pattern *BW1* \
-object inst \
-transitive
```

C.2 query_associate_supply_set

Purpose	Query a previously defined supply set association		
Syntax	<pre>query_associate_supply_set supply_set_ref [-detailed]</pre>		
	supply_set_ref	Specifies the name of the supply set <i>supply_set_ref</i> to query.	
Arguments	-detailed	Returns the supply set association information as a list of { <i>key value</i> } pairs, where <i>key</i> is the name of an argument of the associate_supply_set command (any - prefixes are removed) and <i>value</i> is the value of that argument. Valid keys are supply_set_ref and handle .	
Return value	 There are two distinct return structures. a) If -detailed is not specified, then the supply set association shall be returned in the format o the corresponding associate_supply_set command. b) If -detailed is specified, then the supply set association shall be returned as {key value} pairs. 		

The **query_associate_supply_set** commands queries the association between a supply set and a domain or strategy.

If a supply set is associated with a domain using the following associate_supply_set command:

```
associate_supply_set some_supply_set
    -handle U1/PD1.mem ss
```

then query_associate_supply_set some_supply_set returns the corresponding **associate_supply_set** command as previously defined. If the **-detailed** option is specified the association shall be returned as {*key value*} pairs, i.e.,

```
{supply_set_ref some_supply_set} {handle U1/PD1.mem_ss}
```

Syntax example:

query_associate_supply_set some_supply_set

C.3 query_bind_checker

Purpose	Query a previously defined checker module			
Syntax	<pre>query_bind_checker instance_name [-detailed]</pre>			
	<i>instance_name</i> Specifies the <i>instance_name</i> of the checker module to query. If * is specified, then all checker modules defined shall be returned.			
Arguments	-detailed Returns the checker information as a list of { <i>key value</i> } pairs, where <i>key</i> is the name of an argument of the bind_checker command (any - prefixes are removed) and <i>value</i> is the value of that argument. Valid keys are instance_name , elements , module , bind_to , arch , and ports .			
Return value	 There are three distinct return structures. a) If * is specified for <i>instance_name</i>, then the instance names of all previously defined check modules shall be returned as a Tcl list (a <i>null string</i> shall be returned if no power states a defined). b) If <i>instance_name is</i> specified, then the checker module information for the specified instan shall be returned. c) If -detailed is specified, then the checker module information for the specified instan <i>instance_name</i> shall be returned as {<i>key value</i>} pairs. 			

The **query_bind_checker** command queries any previously defined bind checkers in and below the current scope.

If a bind checker was previously defined as

```
bind_checker chk_p_clks
-module assert_partial_clk
-bind_to aars
-ports {{prt1 clknet2} {port3 net4}}
```

then query_bind_checker chk_p_clks returns the corresponding **bind_checker** command as previously defined. query_bind_checker * returns the instance names of all the previously defined bind checkers, i.e., {chk_p_clks} and if the -detailed option is used, i.e., query_bind_checker chk_p_clks -detailed then the state information is returned as {{instance_name}

chk_p_clks} {elements {}} {module assert_partial_clk} {bind_to aars}
{arch {}} {ports {}}.

It shall be an error if **-detailed** is specified and * is specified for *instance name*.

Syntax example:

query_bind_checker *

C.4 query_cell_instances

Purpose	Query the instances of a mapped cell within the current scope			
Syntax	query_cell_instances cell_name [-domain domain_name]			
Arguments	cell_name	The name of the cell or module to find.		
Arguments	-domain domain_name	Limits the search to the instances in the domain specified.		
Return value	Return a list of the instances that use the named cell or module. The list may be empty.			

The query_cell_instances command can locate all uses of a particular cell.

Syntax example:

```
//To find all instances of a cell named MyCell in the current scope
query cell instances MyCell
```

C.5 query_cell_mapped

Purpose	Query which cell is mapped to this instance		
Syntax	<pre>query_cell_mapped instance_name</pre>		
Arguments	<i>instance_name</i> The name of the instance.		
Return value	Return a cell name.		

The **query_cell_mapped** command can identify the cell that is used for the named instance *instance_name*.

Syntax example:

```
query_cell_mapped top/a/my_inst
```

C.6 query_composite_domain

Purpose	Query a composite domain			
Syntax	query_composite_domain composite_domain_name [-detailed]			
	composite_domain_ nameSpecifies the composite_domain_name to query. If * is specified then the name of all composite domains shall be returned as a Tcl list.			
Arguments	-detailed Returns the composite domain information as a list of {key value} pairs, where key is the name of an argument of the create_composite_domain command (any - prefixes are removed) and value is the value of that argument. Valid keys are composite_domain_name, subdomains, and supply.			
Return value	 There are three distinct return structures. a) If * is specified for <i>composite_domain_name</i>, then all previously defined composite domain shall be returned as a Tcl list (a <i>null string</i> shall be returned if no power states are defined). b) If <i>composite_domain_name</i> is specified (and it is not *), then the composite domain information shall be returned. If the specified <i>composite_domain_name</i> is not a composite domain, but is a non-composite domain, then a 0 shall be returned to indicate this. c) If <i>-detailed</i> is specified, then the composite domain information for the specified <i>composite_domain_name</i> shall be returned as {<i>key value</i>} pairs. 			

The **query_composite_domain** command returns any previously defined composite domains, in and below the current scope. If a composite domain is defined as

```
create_composite_domain dom_combined
  -subdomains {IP1/PDtop IP2/SIM/PD2}
  -supply {primary IP1/PDtop}
```

then query_composite_domain dom_combined returns the composite domain information using the create_composite_domain command previously defined. query_composite_domain * returns all defined composite domains, i.e., {dom_combined}. If the -detailed option is used, i.e., query_composite_domain dom_combined -detailed, then the composite domain information information is returned as {{composite_domain_name__dom_combined}__ {subdomains {IP1/PDtop IP2/SIM/PD2}} {supply {{primary IP1/PDtop}}}.

It shall be an error if **-detailed** is specified and * is specified for *composite_domain_name*.

Syntax example:

query_composite_domain dom_combined

C.7 query_design_attributes

Purpose	Query attributes for an instance or model			
Syntax	<pre>query_design_attributes <-element element_name -model model_name> [-detailed]</pre>			
	-element element_name	A rooted name of instances, named processes, sequential $regs$, or signal names.		
Arguments	-model model_name	A model to query.		
	-detailed	Returns the design attribute information as a list of { <i>key value</i> } pairs, where <i>key</i> is the name of the attribute and <i>value</i> is the value of that attribute. Valid keys are elements , model , and attribute .		
Return value	 There are two distinct return structures. a) If -detailed is not specified, then the attribute information shall be return in the form of the corresponding set_design_attributes command, or a null string shall be returned if no attribute information is defined for the specified <i>element</i> or <i>model</i>. b) If -detailed is specified, then the attribute information for the specified <i>element</i> or <i>model</i> shall be returned as {<i>key value</i>} pairs. 			

The **query_design_attributes** command queries attribute information for a specified *element_name* or *model name*.

For an element that has the following attribute information applied:

```
set_design_attributes -elements lock_cache[0] -attribute {UPF_is_leaf TRUE}
set_design_attributes -elements lock_cache[0] -attribute {UPF_retention
    required}
```

then query_design_attributes -elements lock_cache[0] shall return the attribute information in the form of the corresponding **set_design_attributes** command. The **-detailed** argument shall return the attribute information as *{key value}* pairs, i.e.,

{UPF_is_leaf TRUE} {UPF_retention required}

Syntax example:

query_design_attributes -elements lock_cache[0] -detailed

C.8 query_hdl2upf_vct

Purpose	Query a value conversion table (VCT)			
Syntax	<pre>query_hdl2upf_vct vct_name [-detailed]</pre>			
	vct_nameSpecifies the vct_name to query. If * is specified, then the name of all defined VCTs shall be returned as a Tcl list.			
Arguments	-detailed Returns the VCT information as a list of { <i>key value</i> } pairs, where <i>key</i> is the name of an argument of the create_hdl2upf_vct command (any - prefixes are removed) and <i>value</i> is the value of that argument. Valid keys are vct_name, hdl_type, and table.			
Return value	 There are three distinct return structures. a) If * is specified for vct_name, then all previously defined VCTs shall be returned as a Tcl lis (a null string shall be returned if no VCTs are defined). b) If vct_name is specified (and it is not *), then the VCT information shall be returned in the form of the corresponding create_hdl2upf_vct command. c) If -detailed is specified, then the VCT information for the specified vct_name shall be returned as {key value} pairs. 			

The query_hdl2upf_vct command can list and query any previously defined value conversion table (VCT).

If a VCT specified as

```
create_hdl2upf_vct stdlogic2upf_vss
-hdl_type {vhdl std_logic}
-table {{'U' OFF}
{'X' OFF}
{'O' OFF}
{'1' FULL_ON}
{'2' PARTIAL_ON}
{'Y' OFF}
{'L' OFF}
{'H' FULL_ON}
{'-' OFF}}
```

then query_hdl2upf_vct stdlogic2upf_vss returns the VCT information in the formation of the create_hdl2upf_vct command defined above. query_hdl2upf_vct * returns the defined VCTs, i.e., {stdlogic2upf_vss}, and query_hdl2upf_vct stdlogic2upf_vss -detailed returns the VCT information using {*key value*} pairs, i.e.,

```
{vct_name stdlogic2upf_vss} {hdl_type {vhdl std_logic}} {table {{`U' OFF} {`X'
OFF} {`0' OFF} {`1' FULL_ON} {`Z' PARTIAL_ON} {`W' OFF} {`L' OFF} {`H'
FULL_ON} {`-' OFF}}}
```

It shall be an error if -detailed is specified and * is specified for vct name.

Syntax example:

```
query_hdl2upf_vct stdlogic2upf_vss
```

C.9 query_isolation

Purpose	Query information for an isolation strategy	
Syntax	query_isolation isolation_name -domain ref_domain_name [-detailed]	
Arguments	<i>isolation_name</i> Specifies the isolation strategy to be queried. If * is specified, then a list of isolation strategy names defined for <i>domain_name</i> shall be returned (or a <i>null string</i> if no strategies have been previously defined).	
	-domain ref_domain_nameSpecifies the ref_domain_name for which the isolation strategies are to be queried.	
	-detailed Returns the strategy information as a list of {key value} pairs. Where key is the name of the arguments from the set_isolation command (any - prefixes are removed) and value is the value of that argument. Valid keys are isolation_name, domain, elements, exclude_elements, isolation_power_net, isolation_ground_net, no_isolation, isolation_supply_set, isolation_signal, name_prefix, name_suffix, isolation_sense, clamp_value, sink_off_clamp, source_off_clamp, location, force_isolation, diff_supply_only, and instance.	
	There are three distinct return structures.	
	a) If a * is specified for <i>isolation_name</i> , then a list of the defined isolation strategies for the specified <i>domain name</i> shall be returned.	
Return value	b) If a previously defined isolation strategy is specified for <i>isolation_name</i> and -detailed is not specified, then all arguments of the isolation strategy shall be returned in the format of the corresponding set_isolation command (see <u>6.41</u>).	
	c) If -detailed is specified, then the isolation strategy information shall be returned as a list of <i>{key value}</i> pairs.	

The **query_isolation** command can list the previously defined isolation strategies for the specified power domain *domain name*. All elements returned are referenced to the current scope.

If * is specified for *isolation_name*, then a list of the previously defined isolation strategies for the specified *domain_name* shall be returned. If no strategies are defined then a *null string* shall be returned.

If **-detailed** is specified, then all the parameters of the specified isolation strategy *isolation_name* shall be returned as a Tcl list consisting of {*key value*} pairs. If *value* is a Boolean, then 0 is returned for *False* and 1 is returned for *True*. For example, if the following isolation strategies have been previously defined

```
set_isolation clamp0_strategy
-domain pda
-isolation_supply_set {ISO1 ISO2} -source_off_clamp {0}
set_isolation clamp1_strategy
-domain pda
-isolation supply set {ISO1 ISO2} -clamp 1 -applies to outputs
```

then query_isolation * -domain pda returns {clamp0_strategy clamp1_strategy}. query_isolation clamp0_strategy -domain pda returns the isolation strategy information in the form of the corresponding set_isolation command, as previously defined. query_isolation clamp0_strategy -domain pda -detailed returns

{isolation_name clamp0_strategy} {domain pda} {elements {}} {
 isolation_power_net {}} {isolation_ground_net {}} {no_isolation 0}
 {isolation_supply_set {ISO1 ISO2}} { isolation_signal {}} |{clamp_value
 any} {sink_off_clamp {}} {source_off_clamp 0} {location automatic}
 {force isolation 0}

The following arguments of the **set_isolation** command (see <u>6.41</u>) are not supported by **query_isolation**, as they are expanded on the invocation of the **set_isolation** command:

-applies_to* -source -sink

NOTE—If it is not be possible to return all the strategy information in a single return string, i.e., because of layering, the return information shall be returned as a list of lists. The return value of a detailed query of this form shall be composed as {{detailed_unique_1} {detailed_unique_2} ...}, where each detailed_unique_* shall be an entire detailed query as previously shown.

It shall be an error if

- - detailed is specified and *isolation_name* is *.
- the specified *domain_name* starts with . . or /, i.e., the domain shall be referenced from the current scope, and reside in the current scope or below it.

Syntax example:

query isolation * -domain pda

C.10 query_isolation_control [deprecated]

This is a deprecated command; see also 6.1.

C.11 query_level_shifter

Purpose	Query information for a level-shifter strategy		
Syntax	query_level_shifter level_shifter_name -domain domain_name [-detailed]		
Arguments	level_s	hifter_name	Specifies the level-shifter strategy to be queried. If * is specified then a list of level-shifter strategy names defined for <i>domain_name</i> shall be returned (or a <i>null string</i> if no strategies have been previously defined).
	-domai	in domain_name	Specifies the <i>domain_name</i> for which the level-shifter strategies are to be queried.
	-detail	ed	Returns the parameters of the level-shifter strategy as a list of {key value} pairs, where key is the name of an argument of the set_level_shifter command (any - prefixes are removed) and value is the value of that argument. Valid keys are level_shifter_name, domain, elements, no_shift, threshold, force_shift, rule, location, instance, name_prefix, name_suffix, input_supply_set, output_supply_set, and internal_supply_set.
	There are three distinct return structures.		
Return value	a) If a * is specified for <i>level_shifter_name</i> , then a list of the defined level-shifter strategies the specified <i>domain_name</i> shall be returned.		
	b)	b) If a previously defined level-shifter strategy is specified for <i>level_shifter_name</i> and -deta is not specified, then all arguments of the level-shifter strategy shall be returned in the for of the corresponding set_level_shifter command (see <u>6.43</u>).	
	c)	If -detailed is <i>level_shifter_nar</i>	specified, then the level-shifter strategy information for the specified <i>ne</i> strategy shall be returned as a list of { <i>key value</i> } pairs.

The **query_level_shifter** command can list the previously defined level-shifter strategies and parameters of these strategies. All elements returned are referenced to the current scope.

If a level-shifter strategy is defined as

```
set_level_shifter shift_up
-domain PowerDomainZ
-applies_to outputs
-threshold 0.02
-rule both
```

then query_level_shifter * -domain PowerDomainZ returns all the level-shifter strategies defined for the power domain PowerDomainZ, i.e., {shift_up}. query_level_shifter shift_up -domain PowerDomainZ returns the level-shifter strategy information in the format of the corresponding set_level_shifter command, as previously defined. query_level_shifter shift_up -domain PowerDomainZ -detailed returns the level-shifter information as {key value} pairs, i.e.,

```
{level_shifter_name shift_up} {domain PowerDomainZ} {elements {}} {no_shift 0}
   {threshold 0.02} {force_shift 0} {rule both} {location automatic}
   {name_prefix {}} {name_suffix {}} {input_supply_set {}} {output_supply_set
   {}} {internal_supply_set {}}
```

The following arguments of the set_level_shifter command (see 6.43) are not support by query_level_shifter, as they are expanded on the invocation of the set_level_shifter command:

```
-applies_to <input | output | both>
-source
-sink
```

NOTE—If it is not be possible to return all the strategy information in a single return string, i.e., because of layering, the return information shall be returned as a list of lists. The return value of a detailed query of this form shall be composed as {{detailed_unique_1} {detailed_unique_2} ...}, where each detailed_unique_* shall be an entire detailed query as previously shown.

It shall be an error if

- - detailed is specified and level shifter name is *.
- the specified *domain_name* starts with . . or /, i.e., the domain shall be referenced from the current scope, and reside in the current scope or below it.

Syntax example:

```
query_level_shifter * -domain pda
```

C.12 query_map_isolation_cell [deprecated]

This is a deprecated command; see also 6.1.

C.13 query_map_level_shifter_cell [deprecated]

This is a deprecated command; see also 6.1.

C.14 query_map_power_switch

Purpose	Query the mapping for a switch cell	
Syntax	<pre>query_map_power_switch switch_name [-detailed]</pre>	
Arguments	<i>switch_name</i> Specifies the switch <i>switch_name</i> to query. If * is specified, then the name of all switches shall be returned as a Tcl list.	
	-detailed Returns the switch information as a list of { <i>key value</i> } pairs, where <i>key</i> is the name of an argument of the map_power_switch command (any - prefixes are removed) and <i>value</i> is the value of that argument. Valid keys are switch_name , lib_cells , and port_map .	
Return value	 There are three distinct return structures. a) If * is specified for <i>switch_name</i>, then all previously defined switches shall be returned as a Tcl list (a <i>null string</i> shall be returned if no switches are defined). b) If <i>switch_name</i> is specified (and it is not *), then the switch information shall be returned in the form of the corresponding map_power_switch command. c) If -detailed is specified, then the switch information for the specified <i>switch_name</i> shall be returned as {<i>key value</i>} pairs. 	

The query_map_power_switch command can query the mapping specification for a switch cell.

If a switch cell has a mapping specification defined as

```
map_power_switch switch_swl -lib_cells test_model -port_map {{test_port
      control port test}}
```

then query_map_power_switch * returns all defined switches, i.e., {switch_sw1}. query_map_power_switch switch_sw1 returns the switch mapping information in the format of the corresponding **map_power_switch** command, as previously defined. query_map_power_switch switch sw1 -detailed returns the mapping specification as a list of {key value} pairs, i.e.,

{switch_name switch_sw1} {lib_cells {test_model}} {port_map {}}

It shall be an error if

- *switch name* is * and **-detailed** is specified.
- *switch name* is not a valid switch.

Syntax example:

query_map_power_switch switch_sw1 -detailed

C.15 query_map_retention_cell

Purpose	Query the mapping for a retention strategy		
Syntax	<pre>query_map_retention_cell retention_name_list -domain domain_name [-detailed]</pre>		
	retention_name_list	Specifies the retention strategy name <i>retention_name_list</i> to query. If * is specified, then the name of all retention strategies shall be returned as a Tcl list.	
Arguments	-domain domain_name	The <i>domain_name</i> for which the strategies are to be queried.	
Arguments	-detailed	Returns the strategy information as a list of { <i>key value</i> } pairs, where <i>key</i> is the name of the arguments from the map_retention_cell command (any - prefixes are removed) and <i>value</i> is the value of that argument. Valid keys are retention_strategy , domain , lib_cells , lib_cell_type , lib_model_name , port , and elements .	
	There are three distinct retu	irn structures.	
Return value	a) If * is specified for <i>retention_name_list</i> , then all previously defined retention strategy names shall be returned as a Tcl list (a <i>null string</i> shall be returned if no retention strategies are defined).		
	b) If <i>retention_name</i> returned in the for	<i>_list</i> is specified (and it is not *), then the strategy information shall be m of the corresponding map_retention_cell command.	
	c) If -detailed is retention_name_li.	specified, then the retention strategy information for the specified <i>st</i> shall be returned as { <i>key value</i> } pairs.	

The query map retention cell can query the mapping specification for a retention strategy.

Give a retention mapping specification defined as

map_retention_cell test_PdA -domain {PdA} -elements {foo/U1 foo/U2}

then query_map_retention_cell * -domain PdA returns all the retention strategies defined on PdA, i.e., {test_PdA}.query_map_retention_cell test_PdA -domain PdA returns the retention mapping information in the format of the corresponding **map_retention_cell** command, as previously defined. query_map_retention_cell test_PdA -domain PdA -detailed returns the mapping information as {*key value*} pairs, i.e.,

{retention_strategy test_PdA} {domain PdA} {elements {foo/U1 foo/U2} {model
 {}} {map {}}

NOTE—If multiple mapping specification are defined for different instances of a retention strategy, then multiple **map_retention_cell** commands shall be returned if **-detailed** is not specified, and if **-detailed** is specified, then a list of list of {key value} pairs shall be returned, e.g., {{mapping_specification_1}} {mapping_specification_2}}.

It shall be an error if

- retention strategy is * and -detailed is specified.
- *retention_strategy* is not a valid retention strategy.

Syntax example:

```
query map retention cell * -domain PD1
```

C.16 query_name_format

Purpose	Query information on the name formatting rules		
Syntax	query_name_format [-isolation_prefix -isolation_suffix -level_shift_prefix -level_shift_suffix -implicit_supply_prefix -implicit_supply_suffix -implicit_logic_prefix -implicit_logic_suffix -detailed]		
	-isolation_prefix	Returns the isolation instance and net prefix.	
	-isolation_suffix	Returns the isolation instance and net suffix.	
	-level_shift_prefix	Returns the level-shifter instance and net prefix.	
	-level_shift_suffix	Returns the level-shifter instance and net suffix.	
Augumente	-implicit_supply_prefix	Returns the implicitly created supply net and port prefix.	
7 in guments	-implicit_supply_suffix	Returns the implicitly created supply net and port suffix.	
	-implicit_logic_prefix	Returns the implicitly created logic net and port prefix.	
	-implicit_logic_suffix	Returns the implicitly created logic net and port suffix.	
	-detailed	Returns the parameters of the name format as a list of { <i>key value</i> } pairs, where <i>key</i> is the name of an argument of the query (any - prefixes are removed) and <i>value</i> is the value of that argument.	
Return value	Return the queried parameter if an optional argument is specified or all of the parameters of the name_format command (see $\underline{6.35}$) if none of the optional arguments are specified.		

The query_name_format command lists the current name format rules in effect.

-detailed returns all the name format parameters as a Tcl list consisting of {key value} pairs. For example, if -isolation_prefix is set to ISO_ and -level_shift_prefix is set to LS_, the -detailed option returns the following:

```
{{isolation_prefix ISO_} {isolation_suffix _UPF_ISO} {level_shift_prefix LS_}
    {level_shift_suffix _UPF_LS} {implicit_supply_prefix ""}
    {implicit_supply_suffix _UPF_IS} {implicit_logic_prefix ""}
    {implicit_logic_suffix _UPF_IL}}
```

Syntax example:

```
query_name_format \
-isolation_suffix
```

C.17 query_net_ports

Purpose	Return ports logically connected to a net		
Syntax	<pre>query_net_ports net_name [-transitive [<true false="" ="">]] [-leaf]</true></pre>		
	net_name	The net for which the connected ports are to be listed. Any port connected to <i>net_name</i> in the current scope is returned.	
Arguments	-transitive [<true <br="">FALSE>]</true>	If -transitive is not specified at all, the default is -transitive FALSE . If -transitive is specified without a value, the default value is TRUE .	
	-leaf	Only returns leaf-cell ports connected to <i>net_name</i> . By default, both non-leaf and leaf-cell ports shall be returned.	
Return value	Return a list of ports that are logically connected to the specified net. If no ports are connected, a <i>null string</i> is returned.		

The query_net_ports command lists all ports that are logically connected to a specified net.

-transitive returns all ports that are connected hierarchically to this net (and that are visible within the current scope); otherwise, only ports connected at the scope of *net_name* are returned.

The following conditions also apply:

- The specified *net_name* cannot start with .. or /, i.e., the net shall be referenced from the current scope, and reside in the current scope or below it.
- All ports returned are referenced to the current scope.

Syntax example:

```
query net ports top/a/b/c -transitive
```

C.18 query_partial_on_translation

Purpose	Return the translation of PARTIAL_ON for named tools
Syntax	query_partial_on_translation
Return value	Return the current setting of the translation as a list of { <i>key value</i> } pairs, where <i>key</i> is the name of an argument of the set_partial_on_translation command (any – prefixes are removed) and <i>value</i> is the value of that argument. Valid keys are partial_on_translation , full_on_tools , and off_tools .

The query_partial_on_translation command provides the ability to determine the translation of **PARTIAL_ON** to **FULL_ON** or **OFF**.

The query returns the translation settings as {key value} pairs. If the translation settings are specified as

set_partial_on_translation OFF -full_on_tools {power_analysis_tool_name}
 -off_tools {test_simulator}

then query_partial_on_translation shall return the settings as {key value} pairs of the form

{partial_on_translation OFF} {full_on_tools {power_analysis_tool_name}}
 {off_tools {test_simulator}}

Syntax example:

query partial on translation

C.19 query_pin_related_supply [deprecated]

This is a deprecated command; see also 6.1.

C.20 query_port_attributes

Purpose	Query the port attributes for a specified port	
Syntax	<pre>query_port_attributes port [-detailed]</pre>	
Arguments	<i>port</i> Specifies the port to query.	
	-detailed Returns the power attributes of the port as a list of {key value} pairs, where key is the name of an argument of the set_port_attributes command (any - prefixes are removed) and value is the value of that argument. Valid keys ports, exclude_ports, domains, exclude_domains, elements, exclude_elements, model, attribute, sink_off_clamp, source_off_clamp, receiver_supply, driver_supply, related_power_port, related_ground_port, related_bias_ports, repeater_supply, and pg_type.	
Return value	 There are two distinct return structures. a) If -detailed is not specified, then all attributes of the port port_name shall be returned in the format of the corresponding set_port_attributes command (see <u>6.46</u>). b) If -detailed is specified, then the port attribute information for the specified shall be returned as a list of {<i>key value</i>} pairs. 	

The query_port_attributes command queries the port attribute information for a specified port.

If a port has the following attribute specification

set_port_attributes -ports {A B} -sink_off_clamp 0

then query_port_attributes A returns the attribute information in the form of the corresponding **set_port_attributes** command, as previously defined. query_port_attributes {A} -detailed returns the attribute information as {*key value*} pairs, i.e.,

{port A} {model {}} {sink_off_clamp 0} {source_off_clamp {}} {receiver_supply
 {}} {driver_supply {}}

Syntax example:

```
query_port_attributes B
```

C.21 query_port_direction

Purpose	Return the direction of the specified port	
Syntax	<pre>query_port_direction port</pre>	
Arguments	<i>port</i> The name of the port for which the direction is being queried.	
Return value	Return in , out , or inout .	

The **query_port_direction** command returns the direction of the specified port. The port can be a signal port or a supply port.

The specified *port* cannot start with .. or /, i.e., the port shall be referenced from the current scope, and reside in the current scope or below it.

Syntax example:

```
query_port_direction {top/a/b}
```

C.22 query_port_net

Purpose	Return the net logically connected to a port	
Syntax	<pre>query_port_net port_name -conn <low high="" =""></low></pre>	
Arguments	port_name	The port where this net is to be returned.
	-conn <low high="" =""></low>	Returns the LowConn or HighConn (the default) connection. This option can only be specified if <i>port_name</i> is not on a leaf cell.
Return value	Return the name of the net connected to the specified <i>port_name</i> . If no net is connected, a <i>null string</i> is returned.	

The **query_port_net** command returns the net connected to a specified port (if such a connection exists). A hierarchal port can have both a LowConn and HighConn, so the **-conn** option can be used to specify the net name to return. If no net is connected to the specified port, a *null string* is returned.

The following conditions also apply:

- The specified *port_name* cannot start with .. or /, i.e., the port needs to be referenced from the current scope, and reside in the current scope or below it.
- The returned net is referenced to the current scope.

Syntax example:

query_port_net top/a/b -conn low
C.23 query_port_state

Purpose	Return the state information for a specified port		
Syntax	query_port_state port_name -state state_name [-detailed]		
Arguments	port_name	Simple or hierarchical name of a supply port for which the power state information is to be queried.	
	-state state_name	The <i>state_name</i> being queried. If * is specified, then state information for all states defined for <i>port_name</i> shall be returned.	
	-detailed	Returns the port state information as a list of { <i>key value</i> } pairs, where valid keys are port_name , state_name , and state .	
	There are three distinct return structures.		
D (a) If -state is not specified, then a list of all defined states for the <i>port_name</i> shall be returned as a Tcl list. A <i>null string</i> shall be returned if no states are defined.		
Return value	b) If a <i>state_nam</i> using the corr for all states s	If a <i>state_name</i> is specified, then the state information for the specified state shall be returned using the corresponding add_port_state command. If * is specified, then state information for all states shall be returned.	
	c) If -detailed is	If -detailed is specified, then the state information shall be returned as {key value} pairs.	

The **query_port_state** command lists the previously defined states for *port_name*. If *state_name* is not specified, then a list of defined states for the port shall be returned. If *state_name* is defined, then all parameters of the specified state shall be returned.

-detailed returns all the parameters of *state_name* as a Tcl list consisting of {*key value*} pairs. For example, if a state called active_state is defined on the port VN1 with the state information {0.88 0.90 0.92}, then -detailed option returns the following:

{port name VN1} {state name active state} {state {0.88 0.90 0.92}}

Without the **-detailed** option, the format of the returned parameters shall be in the format of the corresponding **add_port_state** command, i.e.,

add_port_state VN1 -state {active_state {0.88 0.90 0.92}}

It shall be an error if **-detailed** is specified and * is specified for *state name*.

Syntax example:

query_port_state VN1

C.24 query_power_domain

Purpose	Query a power domain	
Syntax	query_power_domain domain_name [-non_leaf -all -no_elements] [-detailed]	
Arguments	domain_name	Specifies the power domain to query. If * is specified, then the name of all defined power domains shall be returned as a Tcl list.
	-non_leaf -all -no_elements	Allows filtering or exclusion for any elements from being returned by the query. The default is to return the non-leaf cells attached to the domain only.
	-detailed	Returns the domain information as a list of { <i>key value</i> } pairs, where <i>key</i> is the name of an argument of the create_power_domain command (any - prefixes are removed) and <i>value</i> is the value of that argument. Valid keys are domain_name , simulation_only , elements , include_scope , supply , scope , and define_func_type .
	There are three distinct re	turn structures.
Return value	a) If * is specified for <i>domain_name</i> , then all previously defined domains shall be returned as a Tcl list (a <i>null string</i> shall be returned if no domains are defined).	
	b) If <i>domain_name</i> is specified (and it is not *), then the domain information shall be returned in the format of the corresponding create_power_domain command.	
	c) If -detailed is sp returned as { <i>key</i>	becified, then the domain information for the specified <i>domain_name</i> shall be <i>value</i> } pairs.

The **query_power_domain** command queries the parameters of a power domain. The **-no_elements** option prevents the elements attached to the domain from being returned by the query command.

If a power domain is created as follows:

```
create_power_domain PD1 -elements {top/U1}
-supply {primary PD1_Primary}
-supply {isolation PD1_ret}
-supply {retention PD1_ret}
-supply {mem_array PD1_ma}
```

then query_power_domain * returns any power domains defined in and below the current scope, i.e., {PD1}. query_power_domain PD1 returns the power-domain information in the format of the corresponding create_power_domain command, as previously defined. query_power_domain PD1 -detailed returns the power-domain information as {key value} pairs, i.e.,

{domain_name PD1} {elements {top/U1}} {supply {{primary PD1_Primary}
 {isolation PD1_ret} {retention PD1_ret} {mem_array PD1_ma}}}

It shall be an error if -detailed is specified and * is specified for domain name.

Syntax example:

query_power_domain PD1 -no_elements -detailed

C.25 query_power_domain_element

Purpose	Return the domain membership information for an instance	
Syntax	<pre>query_power_domain_element design_element</pre>	
Arguments	<i>design_element</i> The design_element for which the domain membership information is to be returned.	
Return value	Return the domain for the specified <i>design_element</i> . If no domain is found, a <i>null string</i> is returned.	

The query_power_domain_element returns the domain membership of the specified object.

The following conditions also apply:

- The specified *design_element* cannot start with .. or /, i.e., the object shall be referenced from the current scope, and reside in the current scope or below it.
- The returned domain is referenced to the current scope.

Syntax example:

query_power_domain_element top/a/b

NOTE—Nets are propagated as necessary through the descendant subtree and may be renamed to avoid name collision; therefore, the same simple name in different scopes may refer to nets that are independent and unconnected.

C.26 query_power_state

Purpose	Return the state information for a power domain or supply set	
Syntax	query_power_state object_name -state state_name [-detailed]	
	object_name	Simple name of a power domain or supply set.
	-state state_name	<i>state_name</i> is the simple name of the state being queried. If * is specified, then state information for all states are returned.
Arguments	-detailed	Returns the power state information as list of { <i>key value</i> } pairs, where <i>key</i> is the name of an argument of the add_power_state command (any - prefixes are removed) and <i>value</i> is the value of that argument. Valid keys are object_name , state_name , supply_expr , logic_expr , simstate , legal , and illegal .
Return value	 There are three distinct return structures. a) If -state is not specified, a list of defined power states for <i>object_name</i> shall be returned as a Tcl list (a <i>null string</i> shall be returned if no power states are defined). b) If <i>state_name</i> is specified, then the state information for this state shall be returned in the format of the corresponding add_power_state command. c) If -detailed is specified, then the power state information shall be returned as {<i>key value</i>} pairs. 	

The **query_power_state** command lists the previously defined power states for the specified *object_name*, which can be a power domain or a supply set. If *state_name* is not specified, then a list of defined states for

object_name shall be returned. If a *state_name* is defined, then all parameters of the specified state shall be returned.

-detailed returns all the parameters of the specified power state *state_name* as a Tcl list consisting of {*key value*} pairs. For example, if a legal state called LPS on the supply set PDA_SUPPLY has the -supply_expr condition {power == `{FULL_ON, 0.8}} and the -logic_expr condition {u1/PdA == GO MODE}, then the -detailed option returns the following:

{state_name LPS} {object_name PDA_SUPPLY} {supply_expr {power == `{FULL_ON, 0.8}}} {logic expr {u1/PdA == GO MODE}} {legal 1} {illegal 0} {simstate {}}

Without the **-detailed** option, the format of the returned parameters shall be in the format of the corresponding **add_power_state** command, i.e.,

```
add_power_state PDA_RET
   -state {LPS
   -supply_expr {power == `{FULL_ON, 0.8}}
   -logic_expr {u1/PdA == GO_MODE}
   -legal}
```

It shall be an error if **-detailed** is specified and * is specified for *state_name*.

Syntax example:

query power state PDA RET -detailed

Purpose	Query the information for a UPF power switch	
Syntax	<pre>query_power_switch switch_name [-detailed]</pre>	
	<i>switch_name</i> Specifies the power switch to query. If * is specified, then the name of all power switches shall be returned as a Tcl list.	
Arguments	-detailed Returns the power-switch information as a list of {key value} pairs, where key is the name of an argument of the create_power_switch command (any - prefixes are removed) and value is the value of that argument. Valid keys are switch_name, domain, output_supply_port, input_supply_port, control_port, on_state, off_state, supply_set, on_partial_state, ack_port, ack_delay, and error_state.	
Return value	 There are three distinct return structures. a) If * is specified for <i>switch_name</i>, then all previously defined switches shall be returned as a Tcl list (a <i>null string</i> shall be returned if no domains are defined). b) If <i>switch_name</i> is specified (and it is not *), then the switch information shall be returned. c) If -detailed is specified, then the switch information for the specified <i>switch_name</i> shall be returned as {<i>key value</i>} pairs. 	

C.27 query_power_switch

The **query_power_switch** command queries the parameters for a UPF defined power switch.

If a power switch is defined as

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

then query_power_switch * returns the name of any switches defined in and below the current scope. query_power_switch sw1 returns the switch information in the format of the corresponding create_power_switch command, as previously defined. query_power_switch sw1 -detailed returns the switch information as a list of {key value} pairs, i.e.,

```
{switch_name sw1} {domain {}} {output_supply_port {vout VN3}}
    {input_supply_port {vin1 VN1} {vin2 VN2}} {control_port {{ctrl_small ON1}
    {ctrl_large ON2} {ss SUPPLY_SELECT}}} {on_state {{full_s1 vin1 {ctrl_small
    & ctrl_large & ss}} {full_s2 vin2 {ctrl_small & ctrl_large & !ss}}}
    {off_state {not_required {~ctrl_small | ctrl_large | ss}} {supply_set {}}
    {on_partial_state {{partial_s1 vin1 {ctrl_small & !ctrl_large & ss}}
    {partial_s2 vin2 {ctrl_small & !ctrl_large & !ss}}} {ack_delay {}}
    {error_state {{no_small {!ctrl_small & ctrl_large}}}}
```

It shall be an error if **-detailed** is specified and * is specified for *switch_name*.

Syntax example:

query_power_switch *

C.28 query_pst [legacy]

Purpose	Query a power state table (PST)	
Syntax	query_pst table_name [-detailed]	
	table_name	Specifies the PST name to query. If * is specified, then the name of all PSTs shall be returned as a Tcl list.
Arguments	-detailed	Returns the pst information as a list of { <i>key value</i> } pairs, where <i>key</i> is the name of an argument of the create_pst command (any - prefixes are removed) and <i>value</i> is the value of that argument. Valid keys are table_name and supplies .
Return value	 There are three distinct return structures. a) If * is specified for <i>table_name</i>, then all previously defined PSTs shall be returned as a Tcl list (a <i>null string</i> shall be returned if no PSTs are defined). b) If <i>table_name</i> is specified (and it is not *), then the table information shall be returned. c) If <i>-detailed</i> is specified, then the table information for the specified <i>table_name</i> shall be returned as {<i>key value</i>} pairs. 	

This is a legacy command; see also 6.1 and 6.19.

The query_pst command queries the information for any defined PSTs.

If a PST is defined as

```
create_pst MyPowerStateTable -supplies {PN1 PN2 SOC/OTC/PN3}
```

then query_pst * returns any defined PSTs, i.e., MyPowerStateTable. query_pst MyPowerStateTable returns the PST information in the form of the create_pst command, as previously defined. query_pst MyPowerStateTable -detailed returns the PST information as a list of {key value} pairs, i.e.,

```
{table name MyPowerStateTable} {supplies {PN1 PN2 SOC/OTC/PN3}}
```

It shall be an error if **-detailed** is specified and * is specified for *table name*.

Syntax example:

query_pst *

C.29 query_pst_state [legacy]

Purpose	Return the state information for a power domain or supply set		
Syntax	query_pst_state state_name -pst table_name [-detailed]		
	state_name	Specifies the name of the state or * for all states.	
	<pre>-pst table_name</pre>	The PST for which the state information is to be queried.	
Arguments	-detailed	Returns the power state information as list of { <i>key value</i> } pairs, where <i>key</i> is the name of an argument of the add_pst_state command (any - prefixes are removed) and <i>value</i> is the value of that argument. Valid keys are state_name , pst , and state .	
	There are three distinct return structures.		
Return value	a) If * is specified for <i>state_name</i> , then all states defined for the specified PST <i>table_name</i> shall be returned as a Tcl list (a <i>null string</i> shall be returned if no power states are defined).		
	b) If <i>state_name</i> the format of	b) If <i>state_name</i> is specified, then the state information for the specified state shall be returned in the format of the corresponding add_pst_state command.	
	c) If -detailed is	s specified, the power state information shall be returned as {key value} pairs.	

This is a legacy command; see also 6.1 and 6.5.

The **query_pst_state** command lists the previously defined power states for *table_name*. If * is specified for the *state_name*, then a list of defined state names shall be returned as a Tcl list. If a *state_name* is defined (and is not *), then all parameters of the specified state shall be returned.

-detailed returns all the parameters of the specified power state *state_name* as a Tcl list consisting of {*key value*} pairs. If a PST is defined as

create_pst pt -supplies { PN1 PN2 SOC/OTC/PN3 }
add_pst_state s1 -pst pt -state { s08 s08 s08 }
add_pst_state s2 -pst pt -state { s08 s08 off }
add pst state s3 -pst pt -state { s08 s09 off }

then query_pst_state * -pst pt returns all the specified states, i.e., {s1 s2 s3}. If the -detailed option is used, i.e., query_pst_state s1 -pst pt -detailed, then the state information shall be returned as {pst pt} {state name s1} {state {s08 s08 s08}.

NOTE—Without the **-detailed** option, the format of the returned parameters shall be in the format of the corresponding **add_pst_state** command.

It shall be an error if **-detailed** is specified and * is specified for *state name*.

Syntax example:

query_pst_state s1 -pst pt -detailed

C.30 query_retention

Purpose	Query the retention strategy information for a domain	
Syntax	query_retention retention_name -domain domain_name [-detailed]	
	retention_name	Specified the retention strategy to be queried. If * is specified, then a list of retention strategy names defined for <i>domain_name</i> shall be returned (or a <i>null string</i> if no strategies have been previously defined).
	-domain domain_name	Specifies the <i>domain_name</i> for which the retention strategies are to be queried.
Arguments	-detailed	Returns the parameters of the retention strategy as a list of { <i>key value</i> } pairs, where <i>key</i> is the name of an argument of the set_retention command (any - prefixes are removed) and <i>value</i> is the value of that argument. Valid keys are retention_name, domain, elements, exclude_elements, retention_power_net, retention_ground_net, retention_supply_set, no_isolation, save_signal, restore_signal, save_condition, restore_condition, retention_condition, use_retention_as_primary, parameters, and instance.
	There are three distinct return structures.	
	a) If a * is specified for <i>retention_name</i> , then a list of the defined retention strategies for the specified <i>domain_name</i> shall be returned.	
Return value	b) If a previously defined retention strategy is specified for <i>retention_name</i> and -detailed specified, then all arguments of the retention strategy shall be returned in the format of the corresponding set_retention command (see <u>6.49</u>).	
	c) If -detailed is <i>retention_name</i> st	specified, then the retention strategy information for the specified rategy shall be returned as a list of { <i>key value</i> } pairs.

The **query_retention** command lists the previously defined retention strategies for the specified power domain *domain_name*. All elements returned are referenced to the current scope.

If * is specified for *retention_name*, then a list of the previously defined retention strategies for the specified *domain_name* shall be returned. If no strategies are defined, then a *null string* shall be returned.

If **-detailed** is specified, then all the parameters of the specified retention strategy *retention_name* shall be returned as a Tcl list consisting of {*key value*} pairs.

If a retention strategy is defined as

```
set_retention my_retention_strategy -domain pda
-retention_supply_set PDA_ret_supply
-save_signal {my_save posedge} -restore_signal {my_restore negedge }
```

then query_retention * -domain pda returns {my_retention_strategy}. query_retention my_retention_strategy -domain pda returns the retention strategy information in the form of the corresponding set_retention command, as previously defined. query_retention my_retention_strategy -domain pda -detailed returns the retention strategy information as a list of {key value} pairs, i.e.,

IEEE Std 1801-2013 IEEE STANDARD FOR DESIGN AND VERIFICATION OF LOW-POWER INTEGRATED CIRCUITS

{retention_name my_retention_strategy} {domain pda} {elements {}}
 {exclude_elements {}} {retention_power_net {}} {retention_ground_net {}}
 {retention_supply_set PDA_ret_supply} {save_signal {my_save posedge}}
 {restore_signal {my_restore negedge}} {save_condition {}}
 {restore condition {}} {output related supply set {}}

NOTE—If it is not be possible to return all the strategy information in a single return string, i.e., because of layering, the return information shall be returned as a list of lists. The return value of a detailed query of this form shall be composed as {{detailed_unique_1} {detailed_unique_2} ...}, where each detailed_unique_* shall be an entire detailed query as previously shown.

It shall be an error if

- - detailed is specified and retention name is *.
- the specified *domain_name* starts with . . or /, i.e., the domain shall be referenced from the current scope, and reside in the current scope or below it.

Syntax example:

query_retention * -domain pda

C.31 query_retention_control [deprecated]

This is a deprecated command; see also 6.1.

C.32 query_retention_elements

Purpose	Query the retention strategy elements	
Syntax	<pre>query_retention_elements retention_list_name [-detailed]</pre>	
Arguments	retention_list_nameSpecifies the retention element group identifier to be queried. If * is specified, then a list of retention element group identifiers shall be returned (or a <i>null string</i> if no groups have been previously defined).	
	-detailed Returns the retention elements as a list of { <i>key value</i> } pairs, where <i>key</i> is the name of an argument of the set_retention_elements command (any - prefixes are removed) and <i>value</i> is the value of that argument. Valid keys are retention_list_name, elements, applies_to, and retention_purpose.	
Return value	 There are three distinct return structures. a) If a * is specified for <i>retention_list_name</i>, then a list of the defined retention group identifiers shall be returned. b) If a previously defined retention group identifier is specified for <i>retention_list_name</i> and -detailed is not specified, then the retention group information shall be returned in the format of the corresponding set_retention_elements command (see <u>6.51</u>). c) If -detailed is specified, then the retention group information for the specified <i>retention_list_name</i> shall be returned as a list of {<i>key value</i>} pairs. 	

The **query_retention_elements** command returns the list of objects that can be used in a **set_retention_elements** command.

If a retention elements definition is

```
set_retention_elements my_retention_group -elements {state_reg}
    -exclude elements {awake from sleep reg}
```

then query_retention_elements * returns all defined retention element groups, i.e., my_retention_group. query_retention_elements my_retention_group returns the retention elements in the form of the set_retention_elements command, as previously defined. query_retention_elements my_retention_group -detailed returns the retention elements as a list of {key value} pairs, i.e.,

```
{retention_list_name my_retention_group} {elements {state_reg}}
    {exclude_elements {awake_from_sleep_reg}}
```

It shall be an error if -detailed is specified and retention list name is *.

Syntax example:

query retention elements my retention group

C.33 query_simstate_behavior

Purpose	Query the simstate behavior information for a domain	
Syntax	query_simstate_behavior -lib name [-model name] [-detailed]	
Arguments	-lib name Sp	becifies the library name.
	-model name Sp	becifies the model name or use * to query all models in the given library.
	-detailed Re thu - p are	eturns the simstate behavior as a list of { <i>key value</i> } pairs, where <i>key</i> is e name of an argument of the set_simstate_behavior command (any orefixes are removed) and <i>value</i> is the value of that argument. Valid keys e simstate_behavior , lib , and model .
Return value	 There are two distinct return structures. a) If -detailed is not specified, then the simstate behavior information shall be returned in the format of a corresponding set_simstate_behavior command. b) If -detailed is specified, then the simstate behavior shall be returned as a list of {key value} pairs. 	

The query_simstate_behavior command queries the simulation simstate behavior for a model or a library.

If a simstate is defined as

set_simstate_behavior ENABLE -lib library1 -model ANDX7_non_power_aware

then query_simstate_behavior -lib library1 -model ANDX7_non_power_aware returns the simstate behavior for the specified model in the format of the corresponding set_simstate_behavior command, as previously defined. query_simstate_behavior -lib library1 -model ANDX7_non_power_aware -detailed returns the simstate information as a list of {key value} pairs, i.e.,

{{lib library1} {simstate behavior ENABLE} {model ANDX7 non power aware}}

If **-model** * is specified, the simstate information shall be returned for all models in the specified library. Because different models can have different simstate behaviors, a list of a list shall be returned for the two behaviors, i.e.,

```
{{simstate_behavior } {lib } {model } {model } ...} {{simstate_behavior }
    {lib } {model } {model } ...}
```

If a simstate is defined as

```
set_simstate_behavior ENABLE -lib library1 -model ANDX7_non_power_aware
set_simstate_behavior DISABLE -lib library1 -model NANDX7_power_aware
```

then a detailed simstate query returns

```
{{simstate_behavior ENABLE} {lib library1} {model ANDX7_non_power_aware}}
{{simstate_behavior DISABLE} {lib library1} {model NANDX7_power_aware}}
```

Syntax example:

query_simstate_behavior -lib library1 -model ANDX7_non_power_aware

C.34 query_state_transition

Purpose	Query a state transition	
Syntax	query_state_transition t -object object_name [-from from_state_lis [-to to_state_list] [-paired {{paired_sta [-legal -illegal] [-detailed]	ransition_name t] tte_list}*}]
	transition_name	Specifies the <i>transition_name</i> to query. If * is specified, then all state transitions for the specified <i>object_name</i> shall be returned as a Tcl list.
	-object object_name	Name of a power domain or supply set for which the state transition information shall be queried.
Arguments	-from from_state_list	If <i>transition_name</i> is *, then <i>from_state_list</i> can be used to filter the returned transitions. A transition name shall only be returned if it starts from any one of the states in <i>from_state_list</i> .
	-to to_state_list	If <i>transition_name</i> is *, then <i>to_state_list</i> can be used to filter the returned transitions. A transition name shall only be returned if it ends at any one of the states in <i>to_state_list</i> .
	<pre>-paired {{paired_state_list}*}</pre>	If <i>transition_name</i> is *, then <i>paired_state_list</i> can be used to filter the returned transitions. A transition name shall only be returned if it ends at any one of the states in <i>paired_state_list</i> .
	-legal -illegal	If <i>transition_name</i> is *, then -legal or -illegal can be specified to restrict the returned transition names. If neither are specified, then both illegal and legal transitions shall be returned.
	-detailed	Returns the transition information as a list of { <i>key value</i> } pairs, where <i>key</i> is the name of an argument of the describe_state_transition command (any - prefixes are removed) and <i>value</i> is the value of that argument. Valid keys are transition_name , object , from , to , paired , legal , and illegal .
	There are three distinct return structures.	
Return value	 a) If * is specified for <i>transition_name</i>, then the name of all transitions for the specified <i>object_name</i> shall be returned as a Tcl list (a <i>null string</i> shall be returned if no state transitions are defined). Returned transition names shall be filtered by the -from, -to, -paired, -illegal, and -legal arguments, if specified. 	
	b) If <i>transition_name</i> returned in the fo	<i>ne</i> is specified (and it is not *), then the state transition information shall be orm of the corresponding describe state transition command
	c) If -detailed is sp returned as { <i>key</i>	becified, then the state information for the specified <i>transition_name</i> shall be <i>value</i> } pairs.

The **query_state_transition** command queries state transition information. All transition states for a specified *object_name* can be queried as a Tcl list if * is specified for *transition_name*. The **-from, -to**, **-paired**, **-legal**, and **-illegal** arguments can be used to filter the returned state transitions when *transition name* is *.

If a state transition is specified as

```
describe_state_transition turn_on -object PdA -from {SLEEP_MODE} -to {GO_MODE}
-paired {DROWSY SLEEP_MODE} -legal
```

IEEE Std 1801-2013 IEEE STANDARD FOR DESIGN AND VERIFICATION OF LOW-POWER INTEGRATED CIRCUITS

then query_state_transition * -object PdA returns a Tcl list of all defined state transitions for PdA, i.e., {turn_on}. query_state_transition * -object PdA -from SLEEP_MODE returns any state transitions starting from the state SLEEP_MODE. query_state_transition turn_on -object PdA -detailed returns the state transition information as a list of {key value} pairs, i.e.,

{transition_name turn_on} {object PdA} {from {SLEEP_MODE}} {to {GO_MODE}}
{paired {{DROWSY SLEEP_MODE}} {legal 1} {illegal 0}

It shall be an error if

- transition name is not * and -from, -to, -paired, -legal, -illegal, or -detailed is specified.
- *transition_name* is not a transition state.

Syntax example:

query_state_transition * -object PdA

C.35 query_supply_net

Purpose	Query a supply net	
Syntax	query_supply_net net_name [-domain domain_name] [-is_supply -detailed]	
	net_name	Specifies the net_name to query. If * is specified, then the name of all supply nets shall be returned as a Tcl list.
	-domain domain_name	Restricts the query to a specified <i>domain_name</i> .
Arguments	-is_supply -detailed	If -is_supply is specified, then a 1 shall be returned if the specified <i>net_name</i> is a supply port and a 0 shall be returned if it is not. If -detailed is specified, then the supply port information shall be returned as a list of { <i>key value</i> } pairs, where <i>key</i> is the name of an argument of the create_supply_net command (any - prefixes are removed) and <i>value</i> is the value of that argument. Valid keys are net_name , domain , and resolve .
	There are four distinct return structures. a) If * is specified for <i>net_name</i> , then all previously defined supply nets shall be returned as a Tablict (a will string shall be astronged if as a supply nets are defined)	
Return value	 b) If <i>net_name</i> is sp c) If -detailed is sp as {<i>key value</i>} pa d) if -is_supply is s otherwise, a 0 sh 	becified (and it is not *), then the net information shall be returned. becified, then the net information for the specified <i>net_name</i> shall be returned birs. becified, then a 1 shall be returned if the specified <i>net_name</i> is a supply port; hall be returned.

The **query_supply_net** command returns the information about a previously created supply net. When called with the **-is_supply** argument, this query can be used to check if the specified *net_name* is a supply net. The **-domain** option restricts the query to the specified *domain name*.

If a supply net is created as follows:

create_supply_net oneh_supply -resolve one_hot

IEEE Std 1801-2013 IEEE STANDARD FOR DESIGN AND VERIFICATION OF LOW-POWER INTEGRATED CIRCUITS

then query_supply_net * returns all supply nets in and below the current scope, i.e., oneh_supply. query_supply_net oneh_supply returns the supply net information in the format of the corresponding **create_supply_net** command, as previously defined. query_supply_net oneh supply -detailed returns the supply net information as a list of {key value} pairs, i.e.,

```
{net_name oneh_supply} {resolve {one_hot}}
```

The following also apply:

- It shall be an error if -detailed, -is_supply, or -supply_set is specified and * is specified for net_name.
- *net_name* is not a supply net unless **-is_supply** is specified.

Syntax example:

query_supply_net add_net -is_supply

C.36 query_supply_port

Purpose	Query a supply port	
Syntax	query_supply_port port_ [- domain domain_nan [-is_supply - detailed	name ne] I]
	port_name	Specifies the <i>port_name</i> to query. If * is specified, then the name of all supply ports shall be returned as a Tcl list. By default, ports are listed on the current scope unless -domain is specified.
	-domain domain_name	Restricts the query to the interface of the specified <i>domain_name</i> .
Arguments	is_supply -detailed	If -is_supply is specified, then a 1 shall be returned if the specified <i>port_name</i> is a supply port and a 0 shall be returned if it is not. If -detailed is specified, then the supply port information shall be returned as a list of <i>{key value}</i> pairs, where <i>key</i> is the name of an argument of the create_supply_port command (any - prefixes are removed) and <i>value</i> is the value of that argument. Valid keys are port_name and direction .
Return value	There are four distinct retu a) If * is specified f Tcl list (a <i>null stri</i> b) If <i>port_name</i> is s c) If -detailed is s returned as { <i>key</i> d) if -is_supply is s port; otherwise, a	urn structures. For <i>port_name</i> , then all previously defined supply ports shall be returned as a <i>ring</i> shall be returned if no supply ports are defined). pecified (and it is not *), then the port information shall be returned. pecified, then the port information for the specified <i>port_name</i> shall be <i>value</i> } pairs. specified, then a 1 shall be returned if the specified <i>port_name</i> is a supply a 0 shall be returned.

The **query_supply_port** command returns the information about a previously created supply port. When called with the **-is_supply** argument, this query can be used to check if the specified *port_name* is a supply port. The **-domain** option restricts the query to the interface of the specified *domain_name*. The interface of a domain in this context is the logic hierarchy boundary between one domain and another, or between a domain and the top-level scope.

If a supply port is created as

create_supply_port VN1 -direction inout

then query_supply_port * returns all supply ports on the current scope, i.e., {VN1}. query_supply_port VN1 returns the supply port information in the format of the corresponding create_supply_port command, as previously defined. query_supply_port VN1 -detailed returns the supply port information as a list of {key value} pairs, i.e.,

{port_name VN1} {direction inout}

The following also apply:

- It shall be an error if **-is_supply** or **-detailed** are specified and *port_name* is *.
- It shall be an error if *port_name* is not * and **-domain** is specified.
- *port_name* is not a supply port unless **-is_supply** is specified.

Syntax example:

query_supply_port jpeg_port -is_supply

C.37 query_supply_set

Purpose	Query a supply set	
Syntax	query_supply_set set_name [-detailed] [-transitive [<true f<="" th="" =""><th>? ALSE>]]</th></true>	? ALSE>]]
	set_name S	pecifies the supply set <i>set_name</i> to query. If * is specified, then the name f all supply sets shall be returned as a Tcl list.
Arguments	-detailed I: a c v r	f -detailed is specified, then the supply set information shall be returned as list of { <i>key value</i> } pairs, where <i>key</i> is the name of an argument of the reate_supply_set command (any - prefixes are removed) and <i>value</i> is the alue of that argument. Valid keys are set_name, function, and efference_gnd.
	-transitive [<true v<br="" ="">FALSE>] d</true>	When -transitive is FALSE (the default), the command applies to the escendants of the elements.
	There are three distinct retur	n structures.
Return	 a) If * is specified for below if -transitive if no supply ports a 	<i>set_name</i> , then all previously defined supply sets in the current scope (and e is specified) shall be returned as a Tcl list (a <i>null string</i> shall be returned re defined).
value	b) If <i>set_name</i> is spec the form of the corr	ified (and it is not *), then the supply set information shall be returned in esponding create_supply_set command.
	c) If -detailed is spec returned as { <i>key val</i>	ified, then the supply set information for the specified <i>set_name</i> shall be <i>lue</i> } pairs.

The query_supply_set commands queries any previously defined supply sets.

If a supply set is created as

create_supply_set relative_always_on_ss

```
-function {power vdd}
-function {ground vss}
-reference_gnd {earth_ground}
```

then query_supply_set * returns the names of any previously created supply sets, i.e., {relative_always_on_ss}. query_supply_set relative_always_on_ss returns the supply set information in the format of the corresponding create_supply_set command, as previously defined. query_supply_set relative_always_on_ss -detailed returns the supply set information using {key value} pairs, i.e.,

```
{set_name relative_always_on_ss} {function {{power vdd} {ground vss}}}
{reference_gnd {earth_ground}}
```

It shall be an error if

- - detailed is specified and *set_name* is *.
- - **transitive** is specified and *set_name* is not *.

Syntax example:

```
query_supply_set relative_always_on_ss
```

C.38 query_upf2hdl_vct

Purpose	Query a value conversion table (VCT)
Syntax	<pre>query_upf2hdl_vct vct_name [-detailed]</pre>
	vct_nameSpecifies the vct_name to query. If * is specified, then the name of all defined VCTs shall be returned as a Tcl list.
Arguments	-detailed Returns the VCT information as a list of { <i>key value</i> } pairs, where <i>key</i> is the name of an argument of the create_upf2hdl_vct command (any - prefixes are removed) and <i>value</i> is the value of that argument. Valid keys are vct_name , hdl_type , and table .
Return value	 There are three distinct return structures. a) If * is specified for vct_name, then all previously defined VCTs shall be returned as a Tcl list (a null string shall be returned if no VCTs are defined). b) If vct_name is specified (and it is not *), then the VCT information shall be returned in the form of the corresponding create_upf2hdl_vct command. c) If -detailed is specified, then the VCT information for the specified vct_name shall be returned as {key value} pairs.

The **query_upf2hdl_vct** command queries can list and query any previously defined value conversion table (VCT).

If a VCT is created as

```
create_upf2hdl_vct upf2vlog_vdd
-hdl_type {sv}
-table {{OFF X} {FULL_ON 1} {PARTIAL_ON 0}}
```

IEEE Std 1801-2013 IEEE STANDARD FOR DESIGN AND VERIFICATION OF LOW-POWER INTEGRATED CIRCUITS

then query_upf2hdl_vct * returns upf2vlog_vdd. query_upf2hdl_vcd upf2vlog_vdd returns the VCT information in the format of the corresponding create_upf2hdl_vct command, as previously defined. query_upf2hdl_vct upf2vlog_vdd -detailed returns the VCT information as {key value} pairs, i.e.,

```
{vct_name upf2vlog_vdd} {hdl_type {sv}} {table {{OFF X} {FULL_ON 1} {PARTIAL_ON
0}}}
```

It shall be an error if **-detailed** is specified and * is specified for vct_name.

Syntax example:

query_upf2hdl_vcd upf2vlog_vdd

Purpose	Query the interface cell in	nformation for a domain
Syntax	query_use_interface_ce -strategy list_of_isolo -domain domain_nam [-detailed]	II interface_implementation_name ation_level_shifter_strategies ne
	interface_implementatio n_name	Specifies the <i>interface_implementation_name</i> to be queried. If * is specified, then a list of interface implementation names shall be returned (or a <i>null string</i> if none have been previously defined).
	-strategy list_of_isolation_level_s hifter_strategies	Specifies the level-shifter or isolation strategy for which the <i>interface_implementation_name</i> is to be queried.
Arguments	-domain domain_name	Specifies the <i>domain_name</i> for which the <i>list_of_isolation_level_shifter_strategies</i> is defined.
	-detailed	Returns the interface cell information as a list of { <i>key value</i> } pairs, where <i>key</i> is the name of an argument of the use_interface_cell command (any - prefixes are removed) and <i>value</i> is the value of that argument. Valid keys are interface_implementation_name , strategy , domain , lib_cells , map , elements , with_clamp , update_any , force_function , and inverter_supply_set
	There are three distinct re	turn structures.
	a) If a * is specifi implementation	ed for <i>interface_implementation_name</i> , then a list of the defined interface identifiers shall be returned.
Return value	b) If a previously d for <i>interface_im</i> , implementation use_interface_c	efined interface implementation identifier is specified <i>plementation_name</i> and -detailed is not specified, then the interface information shall be returned in the format of the corresponding ell command (see <u>6.55</u>).
	c) If -detailed is interface_implement	specified, then the interface implementation information for the specified <i>nentation_name</i> shall be returned as a list of { <i>key value</i> } pairs.

C.39 query_use_interface_cell

The **query_use_interface_cell** command provides the ability to query the interface cell information for a specific *interface_implementation_name*.

If an interface cell is specified as

```
use_interface_cell my_interface -strategy {ISO1 LS1} -domain PD1
    -elements {top/moduleA/port1 top/moduleA/port2 top/moduleA/port3}
    -lib_cells LS_ISO_COMBO
```

then query_use_interface_cell * -domain PD1 -strategy ISO1 returns all the interface cell specifications defined for strategy ISO1 on domain PD1. query_use_interface_cell my_interface -domain PD1 -strategy ISO1 returns the interface cell information in the format of the corresponding **use_interface_cell** command for the strategy ISO1, as previously defined. query_use_interface_cell my_interface -domain PD1 -strategy ISO1 -detailed returns the interface cell information as a list of {key value} pairs, i.e.,

{{interface_implementation_name my_interface} {strategy ISO1} {domain PD1}
 {lib_cells CLASS1} {map {}} {elements {top/moduleA/port1 top/moduleA/port2
 top/moduleA/port3}} {with_clamp {}} {update_any {}} {force_function 0}
 {inverter_supply_set {}}}

It shall be an error if -detailed is specified and interface implementation name is *.

Syntax example:

query_use_interface_cell * -domain PD1 -strategy ISO1

Annex D

(informative)

Replacing deprecated and legacy commands and options

This annex shows the commands and command options that have been categorized as deprecated or legacy since the last version of the standard, and recommendations for replacing them (where applicable).

Legacy constructs (commands and/or options) have not had their syntax and/or semantics updated to be consistent with other commands in this version of the standard, so their descriptions may contain significant obsolete information and their semantics may not be interoperable with the latest UPF concepts. For recommendations on how to use current constructs to replace legacy and deprecated ones, see <u>D.2</u>. For more details on any deprecated constructs, see IEEE Std 1801TM-2009 [B3].

D.1 Deprecated and legacy constructs

The subclause shows any constructs that have been categorized as deprecated or legacy constructs (see also <u>6.1</u>). For recommendations on replacing them, see <u>Table D.1</u>.

D.1.1 Deprecated constructs

This subclause lists the deprecated commands and options.

D.1.1.1 <u>6.2</u>

add_domain_elements domain_name
 -elements element list

D.1.1.2 6.11

connect_supply_net net_name

[-**pins** *pins list*] (This is a deprecated option.) [-**rail_connection** *rail_type*] (This is a deprecated option.)

D.1.1.3 6.17

create_power_domain domain_name

[-include_scope] (This is a deprecated option.)

[-scope instance_name] (This is a deprecated option.)

D.1.1.4 6.30

map_isolation_cell isolation_name
 -domain domain_name
 [-elements element_list]
 [-lib_cells lib_cells_list]
 [-lib_cell_type lib_cell_type]
 [-lib_model_name model_name {-port {port_name net_name}}*]

D.1.1.5 6.31

map_level_shifter_cell level_shifter_strategy
 -domain domain_name
 -lib_cells list
 [-elements element_list]

D.1.1.6 6.32

```
map_power_switch switch_name_list
```

[-domain domain_name] (This is a deprecated option.)

D.1.1.7 6.34

```
merge_power_domains new_domain_name
    -power_domains list
    [-scope instance_name]
    [-all_equivalent]
```

D.1.1.8 6.36

save_upf upf_file_name

[-version string] (This is a deprecated option.)

D.1.1.9 <u>6.41</u>

set_isolation strategy_name

```
[-location <self | other | parent | automatic | fanout | fanin | faninout | sibling>]
fanin | faninout | sibling (These are deprecated options.)
-clamp_value {<0 | 1 | any | Z | latch | value>*}
any (This is a deprecated option.)
-sink_off_clamp {<0 | 1 | any | Z | latch | value> [simstate_list]} (This is a deprecated option.)
-source_off_clamp {<0 | 1 | any | Z | latch | value> [simstate_list]} (This is a deprecated option.)
[-transitive [<TRUE | FALSE>]] (This is a deprecated option.)
```

D.1.1.10 6.42

```
set_isolation_control isolation_name
   -domain domain_name
   -isolation_signal signal_name
   [-isolation_sense <high | low>]
   [-location <self | parent | sibling | fanout | automatic>]
```

D.1.1.11 6.43

set_level_shifter strategy_name

```
[-location <self | other | parent | automatic | fanout | fanin | faninout | sibling>]
fanin | faninout | sibling (These are deprecated options.)
[-transitive [<TRUE | FALSE>]] (This is a deprecated option.)
-threshold <value | list>
list (This is a deprecated option.)
```

D.1.1.12 6.45

set_pin_related_supply library_cell -pins list -related_power_pin supply_pin -related_ground_pin supply_pin

D.1.1.13 6.46

set_port_attributes

[{-domains domain_list [-applies_to <inputs | outputs | both>]}] (This is a deprecated option.) [{-exclude_domains domain_list [-applies_to <inputs | outputs | both>]}] (This is a deprecated option.) [-repeater_supply supply_set_ref] (This is a deprecated option.) [-transitive [<TRUE | FALSE>]] (This is a deprecated option.)

D.1.1.14 6.47

set_power_switch switch_name
 -output_supply_port {port_name [supply_net_name]}
 {-input_supply_port {port_name [supply_net_name]}}*
 {-control_port {port_name}}*
 {-on_state {state_name input_supply_port {boolean_expression}}}*
 [-supply_set supply_set_ref]
 [-on_partial_state {state_name input_supply_port {boolean_expression}}]*
 [-off_state {state_name {boolean_expression}}]*
 [-error_state {state_name {boolean_expression}}]*

D.1.1.15 6.50

set_retention_control retention_name
 -domain domain_name
 -save_signal {{net_name <high | low | posedge | negedge>}}
 -restore_signal {{net_name <high | low | posedge | negedge>}}
 [-assert_r_mutex {{net_name <high | low | posedge | negedge>}}]*
 [-assert_s_mutex {{net_name <high | low | posedge | negedge>}}]*
 [-assert_rs_mutex {{net_name <high | low | posedge | negedge>}}]*

D.1.1.16 6.51

set_retention_elements retention_list_name

[-expand [<TRUE | FALSE>]] (This is a deprecated option.)

D.1.2 Legacy constructs

This subclause lists the legacy commands and options.

D.1.2.1 6.3

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

D.1.2.2 <u>6.5</u>

add_pst_state state_name
 -pst table_name
 -state supply_states

D.1.2.3 6.19

create_pst table_name
 -supplies supply_list

D.1.2.4 6.39

set_domain_supply_net domain_name
 -primary_power_net supply_net_name
 -primary_ground_net supply_net_name

IEEE Std 1801-2013 IEEE STANDARD FOR DESIGN AND VERIFICATION OF LOW-POWER INTEGRATED CIRCUITS

D.1.2.5 6.41

set_isolation strategy_name

[-isolation_power_net net_name] [-isolation_ground_net net_name] (These are legacy options.)

D.1.2.6 6.49

set_retention isolation_name

[-retention_power_net net_name] [-retention_ground_net net_name] (These are legacy options.)

D.2 Recommendations for replacing deprecated and legacy constructs

Table D.1 shows how to use current constructs to replace deprecated and/or legacy constructs.

		-	-	
Command	Options	Recommended command	Recommended options	Reasons for the recommendation
add_domain_elements	domain_name -elements element_list	create_power_domain	domain_name -update	Superseded by a newer command.
add_port_state	port_name -state {name <options>}</options>	add_power_state	object_name -supply_expr boolean_expression	add power state is intended to replace the whole of the PST commands.
add_pst_state	state_name -pst Table_name -state supply_states	add_power_state	-state state_name N/A -supply_expr {boolean_expression}	add power state is intended to replace the whole of the PST commands.
connect_supply_net	-pins pins_list -rail_connection rail_type	N/A connect_supply_net	N/A UPF_pg_type {pg_type_list element_list}	Superseded by the new attribute pg_type.
create_power_domain	-include_scope	create_power_ domain	-elements {.}	Superseded by create_power_domain name
	-scope instance_name	N/A	N/A	-crements {.}- To create a domain in a different scope, use set_scope first and then create_power_ domain.
create_pst	table_name -supplies supply_list	add_power_state	-state state_name N/A -supply_expr {boolean_expression}	add power state is intended to replace the whole of the PST commands.
map_isolation_cell	isolation_name -domain domain_name -elements list -lib_cells list -lib_cell_type type -lib_model_name model -port {port net}	use_interface_cell	-strategy list of_strategies -domain domain_name -elements list -lib_cells list N/A -lib_cells list -force function -map {{port net}}	Superseded by a newer command.

Table D.1—Recommended commands and options for replacing deprecated and legacy constructs

Command	Options	Recommended command	Recommended options	Reasons for the recommendation
map_level_shifter_cell	level_shifter_strategy -domain domain_name -elements list -lib_cells list	use_interface_cell	-strategy list_of_strategies -domain domain_name -elements list -lib_cells list	Superseded by a newer command.
map_power_switch	switch_name_list -domain_domain_name	V/N	N/A	
merge_power_domains	new_domain_name -power_domains list -scope instance -all_equivalent	N/A	N/A	The command is not usable because the semantics are not well defined.
save_upf	upf_file_name -version string	V/N	N/A	
set_domain_supply_net	domain_name -primary_power_net net -primary_ground_net net	associate_supply_set	supply_set -handle supply_set_handle (for both)	Superseded by a more abstract concept.
set_isolation	strategy_name -location fanin faninout sibling -clamp_value any -clamp_value any -sink_off_clamp {<0 1 any [Z latch value> [simstate_list]} -source_off_clamp {<0 1 any [Z latch value> [simstate_list]} -isolation_ground_net net -isolation_ground_net net -transitive [<true false="" ="">]</true>	N/A set_port_attributes set_port_attributes set_isolation N/A	strategy_name N/A -clamp_value any -sink_off_clamp {<0 1 any [Z latch value> [simstate_list]} -source_off_clamp {<0 1 any [Z latch value> [simstate_list]} -isolation_supply_set set (for both) N/A	These -location options are deprecated for clarity of usage. Use set_port_attributes. Constraints are specified with set_port_attributes instead of set_isolation. Constraints are specified with set_port_attributes instead of set_isolation. Superseded by a more abstract concept. -transitive does not make sense for set_isolation or set_level_shifter, because these options are applied at a domain boundary rather than in each

Table D.1—Recommended commands and options for replacing deprecated and legacy constructs (continued)

Command	Options	Recommended command	Recommended options	Reasons for the recommendation
set_isolation_control	isolation_name -domain_domain_name -isolation_signal -isolation_signal -location	set_isolation	isolation_name -domain_domain_name -isolation_signal -isolation_signal -location	Superseded by a newer command.
set_level_shifter	strategy_name -location fanin faninout sibling	N/A	N/A	These -location options are deprecated for clarity of usage. -transitive does not make sense
	-transitive [<true false="" ="">]</true>	N/A	N/A	for set isolation or set level shifter, because these options are applied at a domain boundary rather than in each hierarchical scope.
	-threshold <i>list</i>	N/A	N/A	
set_pin_related_supply	cell -pins list -related_power_pin pin -related_ground_pin pin	set_port_attributes	-model name -ports list -related_power_port port -related_ground_port port	Superseded by a newer command.
set_port_attributes	{-domains <i>domain list</i> [-applies_to <inputs th="" <=""><th>N/A</th><th>N/A</th><th></th></inputs>	N/A	N/A	
	explute bound] {-exclude_domains domain list [-applies_to <inputs outputs="" ="" <br="">both>18</inputs>	N/A	N/A	No way to obtain a wildcard list of domains from which one could be excluded.
	-repeater_supply	N/A	N/A	
	supply set rej -transitive [<true false="" ="">]</true>	N/A	N/A	

Table D.1—Recommended commands and options for replacing deprecated and legacy constructs (continued)

Options	Recommended command	Recommended options	Reasons for the recommendation
<pre>switch_name output_supply_port {port_name [supply_net_name]} {port_name [supply_net_name]} supply_net_name]} {control_port {port_name}} {on_state {state_name input_supply_port {boolean_expression}}]* {boolean_expression}}]* {boolean_expression}}]* {boolean_expression}]]* {error_state {state_name input_supply_port {boolean_expression}}]*</pre>	Υ/N	N/A	Inconsistent with other UPF command semantics.
retention_name -retention_power_net net -retention_ground_net net	set_retention	<i>retention_name</i> -retention_supply_set <i>set</i> (for both)	Superseded by a more abstract concept.
retention_name -domain domain_name -save_signal -restore_signal	set_retention hind_checker	retention_name -domain_domain_name -save_signal -restore_signal	Superseded by a newer command.
-expand [<true false="" ="">]</true>	N/A	N/A	-expand is unclear.
	Options switch name switch name output_supply_port {port_name [port_name]}* {port_name} [supply_net_name]}* {constate {state_name [nput_supply_port {boolean_expression}}}* {boolean_expression}}]* {boolean_expression}}]* {boolean_expression}}]* {boolean_expression}}]* {boolean_expression}}]* poolean_expression}]]* poolean_expression}]]* poolean_expression}]]* poolean_expression}]]* poolean_expression}]]* poolean_expression]]* poolean_expression]]* poolean_expression]]* poolean_expression]]* poolean_expression]]* poolean_expression]]* poolean_expression] poolean_expression] poolean_expression]	OptionsRecommended commandswitch nameN/Aoutput supply_portN/Afoor name [supply_net_name]}foor name [supply_net_name]}foor name [supply_net_name]}foor namesupply_net_name]}foor namesupply_portfoor namesupply_portfoor namefoor namesupply_portfoor namesupply_portfoolean expression}}foolean expression}foolean expression}<	OptionsRecommended commandRecommended optionsswitch name output supply. port foor name [supply, net name]}* [supply_net name]]* [supply_net set pression]]* [supply_net name [supply_net set pression]]* [set extension]]*retention_name [sourd_net net [sourd_net net set extention_name [sourd_net net set extention_name [sourd_net net set retention_name [sourd_net net set extention_name [set retention_name [set extention_name [set extention_name <b< td=""></b<>

Table D.1—Recommended commands and options for replacing deprecated and legacy constructs (continued)

Г

Т

Т

Annex E

(informative)

Low-power design methodology

The purpose of this document is two-fold. First, various design flows with a recommended use model of UPF are illustrated. Second, a simple design example is used to demonstrate how these various power intent aware design flows can be built.

E.1 Design, implementation, and verification flow for a soft IP

In a typical design flow, a soft IP is implemented independently with its own timing, power, and other constraints. Similarly, a power intent file can be created for this soft IP to drive the design, verification, and implementation of its power-management architecture. Consider the simple soft IP shown in Figure E.1.



Figure E.1—Simple soft IP design

This design has the module name top, which contains glue logic at the top, two other blocks (modules modX and modY), and a hard IP (cellA). The hard IP cellA has two sets of power and ground pins with some internal power-management features. (See <u>E.2.1</u> for the discussion on how the hard IP power intent can be created.) The following is a detailed description of the power intent for this soft IP:

- a) Two external supply sets. No external supply ports are defined at the RTL, so the symbolic names, vdd1/vss1 and vdd2/vss2 are used to represent the two external supplies.
- b) The external supply vdd1/vss1 is the primary power/ground (PG) of supply set SS1, which is shared by the hard IP and the rest of the logic of this soft IP. The vddA and vssA ports of the hard IP are also connected to this supply. The supply set SS1 is relatively always on with respect to other supply sets within the scope.

- c) The external supply vdd2/vss2 is the primary PG of supply set SS2, which is the dedicated supply to the hard IP and can be switched off externally. The vddB and vssB ports of the hard IP are also connected to this supply.
- d) There are two power domains in this soft IP: power domain PD1, which contains instance top/i3, and power domain PDTop, which contains all other instances. Both domains can be switched off independently. The primary supply set of the two domains are gated versions of the external supply sets ss1.
- e) There is no separate UPF for the blocks instantiated by I1 and I3.

Figure E.2 illustrates a typical UPF design flow for a soft IP like the one shown in Figure E.1.



Figure E.2—Typical UPF design flow

For each of the three design stages shown in Figure E.2, the design example in Figure E.1 illustrates how the UPF shall be created, used, and passed on to the later stages of the design flow. Starting from RTL design, followed by logic implementation, and then physical implementation, this annex shows how to leverage the configuration UPF (UPF^1 in Figure E.2) to create the power intent with the successive refinement methodology for later stages of the flow. Some best practices on how to create a flow to make sure the configuration UPF can be used at every stage will be discussed as well.

E.2 RTL design stage

The configuration UPF is created at this stage, which typically includes the UPF for the RTL soft IP and the UPF for the hard IPs instantiated within the soft IP. The UPF created at this stage is often referred to as the *configuration UPF*, denoted as UPF^1 in Figure E.2.

E.2.1 UPF modeling for a hard IP

At the RTL stage, a hard IP is typically modeled by a behavioral model, which models the functionality of the cell without the exact modeling of the detailed circuitry inside. In case the behavioral model does not contain a built-in power-aware model or the model is incomplete in terms of describing the power-management features, a UPF model for the internal power structure, as well as power-management control, can be created to enable simulation tools to use UPF simulation semantics to perform power-aware verification. In addition, such a model can also be used by static verification tools to perform structural checks of the RTL design and by implementation tools in the later design stages.

<u>Figure E.3</u> illustrates the internal power structure of the hard IP cellA in the example of <u>Figure E.1</u>. The dotted lines illustrate how some of the boundary ports are connected internally. The solid arrow lines indicate the power supply relationship of each boundary pin.



Figure E.3—Internal power structure of the hard IP

The power intent for the hard IP in Figure E.3 includes the following characteristics:

- a) Two externally defined supply sets: PG supplies vddA/vssA and vddB/vssB.
- b) vddB_int is an internal switched supply controlled by port Y with external power supply vddB.
- c) Input ports W and X are related to vddA/vssA. Internally, port X has internal isolation. As shown in Figure E.3, port X is connected to the data pin of an isolation gate and W is connected to the enable pin of the same isolation gate inside of this hard IP.
- d) Input ports Y and Z are related to vddB/vssB.
- e) Output port Z1 is related to vddB_int/vssB.

f) Output port Z2 is related to vddB/vssB.

Using the macro cell model construct, the example UPF for above hard IP is shown as follows:

```
# Start of hard IP UPF, assume file name is cellA.upf
begin power model upf macroA -for {cellA}
#section 1: define the interfaces of hard IP power model
create power domain PD -elements {.} \
   -supply { SSAH } -supply { SSBH } -supply { SSBH SW }
#section 2: associate the interface supplies to boundary supply ports or
   internally generated supplies
create supply set PD.SSAH -function { power vddA } -function { ground vssA }
   -update
create supply set PD.SSBH -function { power vddB } -function { ground vssB }
   -update
# special handle for internally generated supply
create supply net vddB int
create supply set PD.SSBH SW -function {power vddB int} -function {pround
   vssB} -update
create power switch internal sw -output supply port {out SSBH SW.power} \
-input_supply_port {in vddB} -control_port {ctrl Y} -on_state { ON in !ctrl }
#section 3: define data port and interface supply set handle associations
set_port_attributes -ports { W X } -receiver_supply PD.SSAH
set port attributes -ports { Y Z} -receiver supply PD.SSBH
set port attributes -ports { Z1 } -driver supply PD.SSBH SW
set port attributes -ports { Z2 } -driver supply PD.SSBH
#section 4: define power states for interface supply set handles
add power state PD.SSAH -supply\
-state {ON -simstate NORMAL \setminus
-supply expr {power == { FULL ON 0.7 0.9} && ground == { FULL ON 0}}
-state {OFF -simstate CORRUPT \
-supply expr {power == OFF || ground == OFF}}
add power state PD.SSBH -supply\
-state {ON -simstate NORMAL \
-supply_expr {power == { FULL ON 1.1 1.3} && ground == {FULL ON 0}}
-state {OFF -simstate CORRUPT -supply_expr {power == OFF || ground == OFF}}
add power state PD.SSBH SW -supply\
-state {ON -simstate NORMAL } -supply expr {power == { FULL_ON 1.1 1.3}
   && ground == {FULL ON 0}}
-state {OFF -simstate CORRUPT -supply_expr {power == OFF || ground ==
   OFF } }
#section 5: define system power states of the hard IP
add power state PD -domain\
-state {S1 -logic expr { SSAH == ON && SSBH == ON && SSBH SW == ON }} \
-state {S2 -logic expr { SSAH == ON && SSBH == ON && SSBH SW == OFF }}
-state {S3 -logic expr { SSAH == ON && SSBH == OFF && SSBH SW == OFF }}
-state {S4 -logic expr { SSAH == OFF && SSBH == OFF && SSBH SW == OFF }}
#section 6: Define internal low power logic
set isolation internal iso -domain PD -elements { X } \
-isolation_signal { W } -isolation_sense {low}
#section 7: Define hard IP level isolation constraints
set port attributes -ports {W Z} -clamp value 1
set port attributes -ports {Y} -clamp value 0
end power model
# End of hard IP UPF cellA.upf
```

To demonstrate the methodology of how a UPF can be coded for a hard IP, the example cellA.upf is divided into sections where each section is targeted for the same category of power intent commands. The following subclauses provide a detailed description of each section and some coding guidelines.

E.2.1.1 Power model definition

The command pair **begin_power_model** and **end_power_model** (see <u>6.8</u>) create a definitive boundary for the power intent for the hard IP cell. The created model name is upf_macroA, and it is targeted for the macro cell cellA.

E.2.1.2 Section 1: Define the interfaces of the hard IP power model

The interfaces of the hard IP power model include the top level power domain for this IP and all the supplies of IP. This power domain is also used to specify the system power states at the hard IP level (see <u>E.2.1.6</u> for more details) or some other power intent commands, such as the boundary isolation logic (see <u>E.2.1.8</u> for more details).

In this example, the power domain PD is created to specify the three supply sets of the IP. The supply set handle SSAH is created for the power and ground ports vddA and vssA, and the supply set handle SSBH is created for the power and ground ports vddB and vssB. As indicated in Figure E.3, the hard IP has a data port Z1 related to the internally gated supply of vddB.

As a result, a different interface supply set handle $SSBH_SW$ is created and the **create_power_switch** command is used to describe the internally gated power supply (see <u>E.2.1.3</u>).

E.2.1.3 Section 2: Associate the interface supplies to boundary supply ports or internal gated supplies

In this section, each of the interface supply set handles is associated to specific supply ports or internal supplies of the hard IP by using the commands **create_supply_set** with **-update** option (see <u>6.22</u>). For internal gated supply, the gated supply net is created using **create_supply_net** command (see <u>6.20</u>) and then the **create_power_switch** command (see <u>6.18</u>) is used to define the connection of the internal gated supply.

E.2.1.4 Section 3: Define boundary data port and interface supply set handle associations

In this section, every boundary port of the hard IP that is connected to some logic internally needs to be associated with one of the interface supply handles of the power model. This defines the driver and receiver supplies of hard IP outputs and inputs, respectively, which in turn enables use of those supplies in the **-source** and **-sink** filters of various strategy commands.

This information also enables the verification tool to accurately check the crossing of signals at the integration level to ensure no crossing between two different supply sets is unprotected by an isolation or level-shifter strategy.

E.2.1.5 Section 4: Define power states for interface supply set handles

This section illustrates how to define the power states for each interface supply set handles of the power model. These states will be used to define the system power states of this hard IP in $\underline{E.2.1.6}$.

E.2.1.6 Section 5: Define system power states of the hard IP

The system power state definition of the hard IP, using the power states of interface supply set handles, provides the information on how this IP should be used properly at the block or System on Chip (SoC) level.

The states defined here can be reused by upper-scope UPF files to define the system power states at the integration level, see the integration example in $\underline{E.2.2}$.

E.2.1.7 Section 6: Define internal low-power logic at the boundary

If the IP has some internal special low-power logic around or within the boundary, the information needs to be captured to enable the complete modeling of the hard IP. Such information includes input port isolation, input ports with clamp diodes, floating ports (see <u>Annex G</u>), and feedthrough ports (see <u>Annex G</u>). To specify diode clamps at the input ports of the hard IP, use **define_clamp_diode** (see <u>7.3</u>) command. In this example, the isolation at input port x needs to be modeled, where the other input port W is the control for this internal isolation.

E.2.1.8 Section 7: Define hard IP level isolation constraints

The isolation clamp value constraints at the hard IP inputs indicate when the driver of the input pin, at the design level where the hard IP is instantiated, is switched off what is the expected isolated value. In this example, the isolation value constraints for ports W and Z are logic 1 and the isolation value constraint for port Y is logic 0. Port X has no isolation value constraint since it already has internal isolation.

E.2.2 UPF modeling for the soft IP

The following shows a sample configuration UPF for the RTL design in <u>Figure E.1</u> for enabling power-aware RTL simulation and validation:

```
# Start of top level configuration UPF, assume the file name of top soft.upf
#section 1: load hard IP power models
load upf cellA.upf
# section 2: define the control ports for special low power logic
create logic port sw enl -direction in ;# power switch enable for PDTop
create logic port sw en2 -direction in ;# power switch enable for PD1
create logic port iso en1 -direction in
create_logic_port iso en2 -direction in
create logic port ret en -direction in
#section 3: define power domains and interface supply set handles
create power domain PDTop -elements {.} \
-supply { SSH1 } -supply { SSH2 } ;# interface supply set handles
create power domain PD1 -elements { I3 }
#section 4: integrate hard IP
apply power model upf modelA -scope I2 -supply map { { PD.SSAH PDTop.SSH1 }
   {PD.SSBH PDTop.SSH2} }
#section 5: define power states for supply set handles
add power state PDTop.SSH1 -supply\
-state {ON -simstate NORMAL -supply expr {power == FULL ON && ground ==
   FULL ON}}
-state {OFF -simstate CORRUPT -supply_expr {power == OFF || ground == OFF}}
add power state PDTop.SSH2 \
-state {ON -simstate NORMAL -supply expr {power == FULL ON && ground ==
   FULL ON}}
-state {OFF -simstate CORRUPT -supply expr {power == OFF || ground == OFF}}
add power state PDTop.primary -supply\
-state {ON -simstate NORMAL -supply expr {power == FULL ON && ground ==
   FULL ON } } \
-state {OFF -simstate CORRUPT -logic expr {sw en1}}
add power state PD1.primary -supply\
-state {ON -simstate NORMAL -supply expr {power == FULL ON && ground ==
   FULL ON}} \
```

```
-state {OFF -simstate CORRUPT -logic expr {sw en2}}
#section 6: define system power states of the soft IP
add power state PDTop -domain\
-state {S1 -logic expr \
{ SSH1 == ON && SSH2 == ON && primary == ON && PD1.primary == ON && I2/PD ==
   S1}} \
-state {S2 -logic expr \
{ SSH1 == ON && SSH2 == ON && primary == ON && PD1.primary == OFF && I2/PD ==
   S1}} \
-state {S3 {-logic expr \
{ SSH1 == ON && SSH2 == ON && primary == OFF && PD1.primary == OFF && I2/PD ==
   S1}} \
-state {S4 -logic expr \
{ SSH1 == ON && SSH2 == ON && primary == ON && PD1.primary == ON && I2/PD ==
   S2}} \
-state {S5 -logic expr \
{ SSH1 == ON && SSH2 == ON && primary == ON && PD1.primary == OFF && I2/PD ==
   S2}} \
-state {S6 -logic expr \
{ SSH1 == ON && SSH2 == ON && primary == OFF && PD1.primary == OFF && I2/PD ==
   S2}} \
-state {S7 -logic expr \
{ SSH1 == ON && SSH2 == OFF && primary == OFF && PD1.primary == OFF && I2/PD
   == S3}} \
-state {S8 -logic expr \
{ SSH1 == ON && SSH2 == OFF && primary == OFF && PD1.primary == OFF && I2/PD
   == S4}} \
-state {S9 -logic expr \
{ SSH1 == OFF && SSH2 == OFF && primary == OFF && PD1.primary == OFF && I2/PD
   == S4\}
#section 7: define isolation strategies
set isolation iso1 -domain PDTop -applies_to output \
-isolation supply set PDTop.SSH1 -location self \
-isolation signal iso en1 -isolation sense high -clamp value 0 -
   diff supply only TRUE
set isolation iso2 -domain PD1 -source PD1.primary \
-isolation_supply_set PDTop.SSH1 -location parent \setminus
-isolation signal iso en2 -isolation sense high -clamp value 0 -
   diff supply only TRUE
set isolation iso3 -domain PDTop -source PDTop.primary -sink PDTop.SSH1 \
-isolation supply set PDTop.SSH1 -location self \
-isolation signal iso en1 -isolation sense high -clamp value 1 -
   diff_supply_only TRUE
set isolation iso4 -domain PDTop -source I2/SSBH SW \
-isolation supply set PDTop.SSH2 -location parent \
-isolation signal in2 -isolation sense high -clamp value 1 -diff supply only
   TRUE
#section 8: define retention strategies
set retention ret1 -domain PD1 -retention supply set PDTop.SSH1 \
-save_signal {ret_en negedge } -restore_signal {ret_en posedge}
#section 9: define soft IP level isolation constraint
set port attributes -domain PDTop -applies to inputs -clamp value 0 -
   exclude ports {in3}
set_port_attributes -ports {in3} -clamp_value 1
# end of Top level configuration UPF, top soft.upf
```

To demonstrate the methodology of how a UPF can be coded for a soft IP, the example top_soft.upf is divided into sections where each section is targeted for the same category of power intent commands. The following subclauses provide a detailed description of each section and some coding guidelines.

E.2.2.1 Section 1: Load hard IP power models

If the soft IP instantiates some hard IP with UPF power models, load it at the beginning of the UPF file.

E.2.2.2 Section 2: Define the control ports for special low-power logic

For soft IP, the low-power control signals are typically not available in the golden RTL HDL definitions. In such a case, to enable the power intent description for this soft IP, designers can use the commands shown in this section to declare virtual logic signals that can be used by the strategies or power states in the rest of the file.

E.2.2.3 Section 3: Define power domains and interface supply set handles

In this section, all power domains of this block need to be defined. For RTL power intent specification, the most important information to be defined of each power domain is the extent of the power domain.

For each soft IP UPF, there shall be one and only one power domain declared with **-elements** {.}. This domain is also referred to as the *top-level domain of the block*. This domain has some other significant usage for the rest of power intent specification. First, all interface supply sets incoming to the soft IP or outgoing of the soft IP need to be declared as named supply set handles of this domain. For example, supply set SSH1 and SSH2 are the two interface supplies of this soft IP. These will become the only supply set handles needed at an upper-scope UPF to integrate this soft IP. Second, all system power states of the soft IP will be defined upon this power domain.

Other optional information of a power domain that can be specified for an RTL design includes:

To specify the primary supply set of the power domain, use the **-supply** option. In this example, both power domains PDTop and PD1 are switchable domains whose primary supply set are not defined yet. As a result, the **-supply** option is not used there.

In this example, two power domains are created. PDTop is the top-level power domain and PD1 is the other power domain with only I3 as its extent. In other words, every instance of the soft IP belongs to power domain PDTop except for I3. In the addition, the top-level power domain is specified with two external supply set handles, SSH1 and SSH2, to represent the actually physical supplies incoming to this soft IP that are yet to be defined.

NOTE—In an RTL power intent specification, there is no need to declare the actual supply nets of each supply set in the power intent file. Such information can be updated later as needed, e.g., before the starting of physical implementation.

E.2.2.4 Section 4: Integrate hard IP

In this section, if the soft IP contains any hard IP with a UPF power model loaded, the power model can be instantiated with the soft IP level supply set handle definitions. In this example, the power model upf macroA is instantiated for instance I2, with the following supply set association:

- The hard IP supply set handle SSAH is associated to soft IP supply set PDTop.SSH1.
- The hard IP supply set handle SSBH is associated to soft IP supply set PDTop.SSH2.

For more information, see 6.6.

E.2.2.5 Section 5: Define power states for supply set handles

In this section, the power states of each supply set handle defined for the soft IP are defined. The following is some general coding guidelines for the power state definition of supply set handles:

- In the definition of power states for supply set handle PDTop.SSH1, the supply expression refers to the supply function of the supply set. Therefore, there is no need to prefix the supply function with the supply set handle name.
- PDTop.SSH1 and PDTop.SSH2 are defined for the interface supply set handles of the soft IP.
 Without knowing how the actual supplies behave, each supply set is defined with two power states, one for normal operation mode and one for switched-off mode as indicated by the -simstate option.
- The primary supply set handles of PDTop and PD1 represent the gated supplies of PDTop.SSH1; therefore, to define the OFF state, the logic expression needs to be specified by using the option -logic_expr to describe the condition under which the primary supply set is at this power state. This information is essential for power-aware simulation.

Since the power states defined on each supply set handle are simple, an alternative is to skip the power state specification for each supply set handle by using the **DEFAULT_NORMAL** and **DEFAULT_CORRUPT** state for supply sets (see <u>4.6.3</u>). For example, the power state definitions for PDTop.SSH1 and PDTop.SSH2 can be removed and the two predefined supply set states can be used in Section 6 to describe the system power states for these two supply sets.

E.2.2.6 Section 6: Define system power states of the soft IP

System power states of a soft IP are defined in the top-level power domain. A system power state is specified using the previously defined power states on the supply set handle, power domain, or system power state of another hard IP or soft IP.

Consider the system power state S1 of this example, the soft IP is considered in the power state S1 when all the following clauses are true:

- PDTop.SSH1 is at power state ON
- PDTop.SSH2 is at power state ON
- PDTop.primary is at power state ON
- PD1.primary is at power state ON
- Hard IP 12 is operated in system power state S1

NOTE—To access the system power state of a hard IP, use the hard IP instance name in the logic expression. The prefix for PDTop.SSH1, PDTop.SSH2, and PDTop.primary are abbreviated since the power states are defined upon power domain PDTop.

As stated in E.2.2.5, if the power state for each supply set handle is not defined then the predefined supply set state **DEFAULT_NORMAL** can be used to replace the state ON and **DEFAULT_CORRUPT** can be used to replace the state OFF.

E.2.2.7 Section 7: Define isolation strategies

Isolation strategies need to be specified for all outputs of power domains that can be switched off. Even though the isolation strategy command has many options, for RTL specification only the following information is required:

— Isolation targets: These are the ports requiring isolation. Designers can use various filters such as -applies_to, -source, or -sink to select domain boundary ports for isolation purpose. If the exact port name is known, use the -elements option

- Isolation control: Use -isolation_signal to specify the isolation control signal name, which may be a virtual logic port name declared earlier, and -isolation_sense to indicate the signal is active high or low to enable the isolation functionality.
- *Isolation type*: Use -clamp_value to indicate the isolation output values. It is recommended to specify a known value to ensure consistency between RTL simulation and gate-level implementation.
- Isolation supply: Use -isolation_supply to specify the supply set that will power the isolation logic. This information is important because the isolation supply needs to be verified as a component of the overall isolation functionality. The same information can be used by implementation tools to connect the supplies for isolation and by static checking tools to verify the power connectivity. An alternative to specifying this information is to use the default isolation supply handle of the referenced power domain by this strategy, such as PD1.default_isolation. However, to ensure consistent simulation and implementation semantics, the default isolation supply set handle needs to be resolved into a real supply set or handle by using associate_supply_set.

The following information of the isolation strategy is optional, but recommended:

The option **-diff_supply_only** is recommended when the user does not want to have isolation inserted between signals driven by and driving to the same supply set.

E.2.2.8 Section 8: Define retention strategies

Retention strategies are required only if some RTL registers or flops of a switchable power domain are targeted for retention functionality. Even though the retention strategy command has many options, for RTL specification only the following information is required:

- Retention targets: Use the -elements and -exclude_elements options to select the target sequential instances for retention purpose or, by default, select all sequential instances of the referenced power domain specified in the -domain option. If set_retention_elements was previously used to specify a list of targeted registers, the retention element list name can be directly referenced in -elements.
- Retention control: Use -save_signal and -restore_signal to specify the retention control signal name and its sense. There are different flavors of retention strategies to support different types of retention cells. Designers need to keep in mind that the retention strategy specified is targeted for a specific retention technology cell to be used in implementation.
- Retention supply: Use -retention_supply to specify the supply set that will power the retention logic when the primary supply to the retention logic is switched off. This information is important because the retention supply needs to be verified as a component of the overall retention functionality. The same information can be used by implementation tools to connect the supplies of retention cells and by static checking tools to verify the power connectivity. An alternative to specifying this information is to use the default retention supply handle of the referenced power domain by this strategy, such as PD1.default_retention. If the -default_retention supply is used, UPF requires that it be associated with a user-defined supply set during implementation.

The following information of a retention strategy is optional and only needed to model different variations of retention logic:

- a) The option **-use_retention_as_primary** needs to be specified if the targeted retention technology has its output related to the retention supply set. This option also changes the driving supply information of a signal during the source and destination analysis of an isolation or level-shifter strategy. For a signal driven by the output of a register or flop targeted for retention
 - 1) without the option specified, the source supply set of a signal driven by the specified retention logic is the primary supply set of the retention logic;
 - 2) with the option specified, the source supply set of the signal driven by the specified retention logic is the retention supply set of the retention logic.
b) Use the option **-parameters** to specify variations of the simulation semantics of the targeted retention logic.

The preceding detailed descriptions on each section of the configuration UPF file top_soft.upf provide a good starting point for some of the most commonly used UPF constructs for RTL modeling of low-power design designs.

The following power intent may also be included for configuration UPF, but they are not required:

c) *Level-shifting strategy*: If the driving and receiving logic of a net are powered by supplies that can be at significantly different voltages, a level-shifter will be required where that net crosses a domain boundary. A default level-shifting strategy (see <u>6.43</u>) is applied if no user-defined level-shifting strategy is provided. If a designer chooses to write level-shifting strategies for an RTL design, define the voltage levels for all supplies using **-supply_expr** in **add_power_state** for all supply set power states. If the level-shifter strategies are not specified for an RTL design, they can be appended to the configuration UPF at later design stages, as shown in <u>E.3.1</u>.

The RTL golden power intent may also include the following information, but this is not recommended as these implementation details are meant to be specified only for the implementation of the golden power intent in later design stages, i.e., logical or physical implementation.

- d) *Supply nets*: For an RTL design, supply nets can be represented abstractly using supply sets, which define a collection of supply functions that will eventually be implemented by individual supply nets. Each function of a supply set can be referenced using a supply net handle (see <u>5.3.3.2</u>) where a reference to a supply net is required.
- e) Power/ground switches: For an RTL design, there is no need to define any power-switch strategy. To specify the control conditions for a switchable power domain or supply set, use the -logic_expr expression in the corresponding add_power_state command for the supply set handle. Powerswitch strategy can be added later on, before physical implementation when the power switch is actually implemented.

To understand how to update the golden power intent with supply nets and switches, refer to $\underline{E.4.1}$ for more details.

E.2.2.9 Section 9: Define soft IP level constraints

Since the configuration UPF for a soft IP contains all the implementable power intent, the only constraint useful for the integration of the IP is any isolation constraints of the input ports if they are not already isolated within the IP.

In the example in <u>E.2.2</u>, the first command specifies the isolation values for all input ports default to logic 0, including the virtual ports created within the UPF in <u>E.2.2.1</u>. However, the second command overwrites the isolation constraint for port in 3 to be logic 1. This is allowed since the second command is a more specific command than the first one and, thus, takes precedence (see <u>5.8</u>).

E.2.3 How to use configuration UPF

As illustrated in Figure E.2, at the RTL design stage, designers need to first perform a quality check on the UPF including a "language lint" check to catch syntax and usage errors and perform some design-dependent consistency checks on the power intent. For example, if an isolation strategy is applied to a signal, but the driving and receiving logic of the signal are on and off together in all power states, then the check may issue a warning to identify this isolation strategy as redundant. The designer can then decide on whether this isolation strategy should be part of the power intent. Once the UPF passes the quality check, designers can run RTL simulation to validate the power-aware functionality of the design, such as power-up/-down

sequence, isolation or state retention control sequence, etc. An optional step before power-aware RTL simulation is to create verification plan and metrics to enable advanced verification methodology, such as power state coverage analysis and automatic testbench generation. The final UPF at the end of RTL design stage will be considered as the golden intent and passed on to the next design stage.

<u>Figure E.4</u> shows how isolation logic is inserted based on the golden UPF specification $top_soft.upf$ for the example in <u>Figure E.1</u>. The following subclauses explain how each of the isolation logic is inserted based on the UPF.



Figure E.4—Block example with isolation logic inserted

E.2.3.1 Isolation strategy iso1

This isolation strategy indicates the output ports of PDTop need to be isolated with value 0 and control signal iso_en1. Per definition of the domain PDTop and the semantics of the interface of a domain, output ports out1, out2, out3, and the HighConn side of ports I2/X, I2/Z, and I3/A are the ports targeted by this strategy. In addition, the strategy indicates the location of the logic is **self**; therefore, four isolation logic, at ports out1, out2, out3, and I3/A, are inserted as shown in Figure E.4 for isolation strategy iso1.

Note that no isolation logic is inserted at ports I2/X and I2/Z as there is another more specific isolation strategy iso3 on the same targeted ports. To understand the precedence policy of UPF, see <u>5.8</u>.

E.2.3.2 Isolation strategy iso2

This isolation strategy indicates all ports of PD1 that are also driven by logic powered by supply set PD1.primary need to be isolated with value 0 and control signal iso_en2 . Per definition of the domain PD1 and the semantics of the interface of a domain, the LowConn side of output ports I3/Z1 and I3/Z2 are the ports targeted by this strategy. In addition, the strategy indicates the location of the logic is at the parent of the selected ports; therefore, two isolation cells are inserted in the module Top as shown in the Figure E.4 because of this strategy.

E.2.3.3 Isolation strategy iso3

This isolation strategy targets the HighConn side of ports 12/X and 12/Z for isolation based on the definition of PDTop and the semantics of the interface of a domain. Even though the two ports are also selected by isolation strategy iso1, according to the precedence policy of UPF (see 5.8), the isolation strategy iso3 is more specific and hence takes the precedence over the strategy iso1. In addition, the strategy indicates the location of the logic is self; therefore, two isolation logic are inserted as shown in Figure E.4 for isolation strategy iso3.

E.2.3.4 Isolation strategy iso4

This isolation strategy targets the HighConn side of ports I2/Z1 for isolation based on the definition of PDTop and the semantics of the interface of a domain. Since the strategy indicates the location of the logic is **self**, one isolation logic is inserted as shown in Figure E.4 for isolation strategy iso4. There are two differences need to be pointed between iso4 and iso1/iso3, even though all three strategies are defined on the same reference domain PDTop:

- iso4 is intended for ports driven by supply set of I2/SSBH_SW, which is an internal supply set derived from SSBH within the hard IP (see cellA.upf) or PDTop.SSH2 at the soft IP level due to the supply map in apply_power_model command; while iso1/iso3 are intended for ports driven by the supply set PDTop.primary, which is derived from PDTop.SSH1.
- As a result, the isolation enable signal and the isolation supply set of iso4 are different from that of iso1/iso3.

NOTE—The HighConn side of port 12/Z2 is also an interface of domain PDTop, but it is not covered by any isolation strategy. This causes no problem because the driver supply set of 12/Z2 is SSBH within the hard IP or PDtop.SSH2 at the soft IP level due to the supply map in **apply_power_model**. From the soft IP system power state definitions, PDTop.SSH2 will not be off when PD1.primary is on, so there is no isolation logic required for port 12/Z2.

E.3 Logic implementation

The logic implementation stage includes logic synthesis, Design for Test (DFT) synthesis, and gate-level simulation. The following information is typically required in addition to the power intent specified in the RTL stage:

- a) If a UPF file exists for special low-power cells, load it into the implementation UPF.
- b) If the power-domain voltage information is not specified at in the configuration UPF, specify it in the implementation UPF.
- c) If level-shifters are needed but not specified in the configuration UPF, specify them in the implementation UPF.
- d) If designers have some preferences for specific library cells to be used for state retention, isolation, and level-shifting strategies, specify them in the implementation UPF.

The preceding additional power intent corresponds to the annotation R1 in Figure E.2.

E.3.1 Logic implementation UPF for the soft IP

The configuration UPF (top_soft.upf) described in <u>E.2</u> can be refined further to become the implementation UPF for driving logic synthesis, by adding the following additional commands (typically in a separate file that loads the configuration UPF file first).

- # Start of incrementation implementation UPF, assume the file name of top_impl.upf
- # Section 1: Add voltage information for supply set states

IEEE Std 1801-2013 IEEE STANDARD FOR DESIGN AND VERIFICATION OF LOW-POWER INTEGRATED CIRCUITS

```
add power state PDTop.SSH1 -supply\
-state {ON -supply expr {power== {FULL ON 0.8}} -update}
add power state PDTop.SSH2 -supply\
-state {ON -supply expr {power== {FULL ON 0.8 1.2}} -update}
add power state PDTop.primary -supply\
-state {ON -supply expr {power== {FULL ON 0.8}} -update}
add power state PD1.primary -supply\
-state {ON -supply expr {power== {FULL ON 0.8}} -update}
# Section 2: Add level-shifting strategy
set level shifter lvl1 -domain PDTop -source PDTop.primary -sink PDTop.SSH2 \
-input supply set PDTop.primary -output supply set PDTop.SSH2
set level shifter lv12 -domain PD1 -source PD1.primary -sink PDTop.SSH2
-input supply set PD1.primary -output supply set PDTop.SSH2
# Section 3: Add library info for retention strategy
map retention cell ret1 -domain PD1 -lib cells {my ret cell1 my ret cell2}
# Section 4: Add library info for isolation strategy
use interface cell isol cells -strategy isol -domain PDTop \
-lib_cells {my_iso_cell1}
use interface cell iso1 cells -strategy iso2 -domain PDTop \
-lib cells {my iso cell1}
use interface cell isol cells -strategy iso4 -domain PDTop \
-lib cells {my iso cell12}
# Section 5: Add library info for level-shifting strategy
use interface cell lvl1 cells -strategy lvl1 -domain PDTop \
-lib_cells {my_lvl_cell1 my_lvl_cell2}
use interface cell lvl2 cells -strategy lvl2 -domain PD1 \
-lib cells {my lvl cell1 my lvl cell2}
# section 6: add library info for combined level-shifting and isolation cells
use interface cell enabled lvl -strategy {iso3 lvl1} -domain PDTop \
-lib cells {en lvl}
# end of incrementation implementation UPF top impl.upf
```

This demonstrates what typical information needs to be added to the configuration UPF, i.e., top_soft.upf, to drive logic synthesis. Designers can either add the command source top_impl.upf at the end of the top_soft.upf or add the command source top_soft.upf at the start of top impl.upf or create a new UPF with the following commands:

```
source top_soft.upf
source top impl.upf
```

As shown in Figure E.2, top_soft.upf corresponds to the configuration UPF denoted as UPF¹, top_impl.upf corresponds to the additional implementation information denoted as R1, and the combined version of top_soft.upf and top_impl.upf corresponds to the implementation UPF denoted as UPF² in Figure E.2.

NOTE—Breaking a complete UPF description into configuration UPF and implementation specific UPF is a good methodology to follow but it is not mandatory. Designers can choose to specify all or part of the information in the implementation UPF within the configuration UPF as well. For example, if a designer wants to verify the interaction of voltage changes between domains, the configuration UPF then needs to include the voltage information as shown in Section 1 in top_impl.upf.

In the preceding example, the file top_impl.upf is divided into a few sections where each section is targeted for the same category of power intent. The following subclauses provide a detailed description of each section with some coding guidelines.

E.3.1.1 Section 1: Voltage definitions for supply set states

The power states defined in the configuration UPF top_soft.upf do not contain any voltage information. To describe any level-shifter requirement or the voltages of each power domain for synthesis, there is a need to update each supply set power state with voltage information for each supply net, including any primary power nets, primary ground nets, nwell supply nets, and pwell supply nets.

Section 1 of top_impl.upf demonstrates how the supply net voltage information can be updated to each supply set power state. By default, the ground supply voltage is 0v, the nwell supply voltage is the same as the voltage of primary power, and the pwell supply voltage is the same as the voltage of primary ground. Also note the supply net handle is used in this example instead of the actual supply net name. Therefore, designers can specify the voltages for each supply function of a supply set without the need to create the supply net in the logic implementation UPF.

E.3.1.2 Section 2: Level-shifting strategies

A level-shifter strategy specification typically requires the following information:

Level-shifting targets: These are the ports that require level-shifters. Designers can use various filters such as **-applies_to**, **-source**, and **-sink** to select domain boundary ports that need level-shifters. If the exact port name is known, use the **-elements** option.

The following information of a level-shifter strategy is optional, but recommended:

-input_supply_set/-output_supply_set are recommended when the input supply set and output supply set are not the default source and sink supply set of the port.

In this example, two strategies are specified, one for domain PDTop and one for domain PD1. Each levelshifting strategy specifies that a level-shifter is to be inserted for any output that is driven by the respective domain's primary supply and received by logic that is powered by the other supply (PDTop.SSH2) of the soft IP. These strategies address the potential voltage difference between the primary supplies of PDTop and PD1, both of which are nominally 0.8 V, and that of the supply PDTop.SSH2, which can range from 0.8 V up to 1.2 V.

E.3.1.3 Section 3: Library cell requirements for retention strategy

By default, implementation tools shall automatically select the right retention cells to implement a retention strategy. However, the user can specify the desired retention cells by using **map_retention_cell**.

E.3.1.4 Section 4: Library cell requirements for isolation strategy

By default, implementation tools shall automatically select the right isolation cells to implement a isolation strategy. However, the user can specify the desired isolation cells by using **use_interface_cell**.

E.3.1.5 Section 5: Library cell requirements for level-shifting strategy

By default, implementation tools shall select the right level-shifters to implement a level-shifting strategy. However, the user can specify desired the level-shifter cells by using **use_interface_cell**.

E.3.1.6 Section 6: Library cell requirements for combined isolation and level-shifting strategy

If there are both an isolation strategy and a level-shifting strategy specified on the same port, special lowpower cell such as an enabled level-shifter or a level-shifter isolation combo cell can be used to implement the two strategies together. To specify such requirements, use use_interface_cell by specifying both strategy names as illustrated in Section 6 of top impl.upf.

E.3.2 How to use the logic implementation UPF

The voltage information specified for each supply set power state can be used by verification tools to check which power-domain crossings require a level-shifter. The level-shifter strategies specify the requirements of level-shifter insertion for synthesis tools. Consider the level-shifter strategies in top_impl.upf, there is no domain crossing with a source of PD1.primary and sink of PDTop.SSH2. As a result, there is no level-shifting logic inferred from the strategy lvl2. For the strategy lvl1, the only crossing that matches the specification of the strategy is the crossing from I1/O2 to I2/Z. Based on the definition of PDTop and the semantics of the interface of a domain, the HighConn side of I2/Z is selected for this level-shifter strategy. In Figure E.5, the level-shifter logic to be inferred by synthesis is shown on the crossing. In addition, in E.2.3 the same port I2/Z is also selected by the isolation strategy iso3. Therefore, the command use_interface_cell can be used to direct the synthesis tool to use an enable level-shifter to implement both strategies, as shown by the shaded circle at I1/O2 in Figure E.5.





E.3.3 UPF usage after logic synthesis

After logic synthesis, typical design steps include gate-level simulation and DFT synthesis. As illustrated in Figure E.2, there is a choice of using the implementation UPF for synthesis (UPF^2 in the diagram) or the UPF written out from synthesis tool (UPF^a in the diagram) to drive the post synthesis design steps. The pros and cons of each approach are as follows.

E.3.3.1 Using UPF² for post synthesis

Pros

 Uses the same UPF for RTL simulation and gate level simulation and verification to ensure closedloop verification. — Maintain all human readable configuration UPF specification.

Cons

 Need to make sure the implementation UPF is consistent with the gate-level netlist, in terms of design hierarchies, object names, etc.

E.3.3.2 Using UPF^a for post synthesis

Pros

— UPF is consistent with the gate-level netlist, in terms of design hierarchies, object names, etc.

Cons

- Extra steps need to be involved to make sure the power intent is not changed from the power intent specified in configuration UPF.
- The machine-generated UPF is not human readable, and it is impossible to use the successive refinement methodology for the physical implementation stage as illustrated in E.3.3.3.

From the perspective of the successive refinement methodology, it is recommended to take UPF^1 for post synthesis usage. In this case, designers need to make sure the synthesis stage does not create object name changes that invalidate the design object references in the original UPF^1 .

E.3.3.3 UPF changes after DFT synthesis

DFT synthesis typically creates some new ports and connections in the design that may create new domain crossings that are not covered by the original power intent. Designers need to make sure either of the following occur:

- All newly created ports are covered by existing strategies, which is possible if the strategy was written without using -elements to specify the exact port name. For example, if DFT synthesis created a new crossing from I1 to I3, the strategy isol can still cover this port in terms of isolation requirements.
- A new isolation strategy is added to the original UPF to cover the new crossing before the physical implementation stage.

E.4 Physical implementation

Physical implementation includes all the steps from power planning, placement, routing, power-switch insertion, physical optimization, and sign-off to generating the final physical netlist and layout. The following information is typically required in addition to the power intent specified for the logic implementation stage:

- a) Supply port definitions
- b) Supply net definitions
- c) Supply net associations with supply set functions
- d) Power-switch definitions
- e) Other physical implementation constraints, such as the requirements for repeaters (see 6.48)

The preceding additional power intent corresponds to the annotation R2 in Figure E.2.

E.4.1 Physical implementation UPF for the soft IP

Based on the implementation UPF (top_soft.upf and top_impl.upf) described in <u>E.3.3</u>, the following additional UPF commands can be added on top of the implementation UPF to drive physical implementation:

```
# Start of physical implementation UPF, assume the file name of top phy.upf
# Section 1: define supply ports
create_supply_port vdd1
create_supply_port vss1
create supply port vdd2
create supply port vss2
# Section 2: define supply nets
create supply net vdd top -domain PDTop
create supply net vdd1 sw -domain PD1
# Section 3: associate supply nets to supply sets
create supply set ss1 -function {power vdd1} -function {ground vss1}
associate_supply_set ss1 -handle PDTop.SSH1
create_supply_set ss2 -function {power vdd2} -function {ground vss2}
associate supply set ss2 -handle PDTop.SSH2
create supply set PDTop ss -function {power vdd top} -function {ground vss1}
associate supply set PDTop ss -handle PDTop.primary
create supply set PD1 ss -function {power vdd1 sw} -function {ground vss1}
associate supply set PD1 ss -handle PD1.primary
# Section 4: define power switch strategy
create power switch PDTop switch \
-output supply port { sw out vdd top} \
-input_supply_port {sw_in vdd1} \
-control port {pso sw en1} \
-on state { top on sw in {!pso} } -off state {top off sw in {pso} }
create power switch PD1 switch \
-output supply port { sw out vdd1 sw} \
-input supply port {sw in vdd1} \
-control port {pso sw en2} \
-on state { top on sw in {!pso} } -off state {top off sw in {pso} }
# end of physical implementation UPF top phy.upf
```

This example demonstrates the typical physical information that needs to be added to the implementation UPF to drive physical implementation. Designers can either add the source top_phy.upf command at the end of the UPF file top_impl.upf or add the source top_impl.upf command at the start of the UPF file top impl.upf or create a new UPF file with the following commands:

```
source top_soft.upf
source top_impl.upf
source top phy.upf
```

As shown in <u>Figure E.2</u>, top_phy.upf corresponds to the additional physical information denoted as R2, and the combination of all three UPF files correspond to physical implementation UPF denoted as UPF^3 in Figure E.2.

NOTE—Breaking a complete UPF description into configuration UPF, logic implementation specific UPF, and physical implementation UPF is a good methodology to follow but it is not mandatory. Designers can choose to specify all or part of the information in the physical implementation UPF within the configuration UPF or logic implementation UPF as well. For example, a designer may choose to specify the supply ports and supply nets in top_soft.upf or top_impl.upf.

The top_phy.upf file is divided into a few sections where each section is targeted for the same category of power intent. The following provides a detailed description of each section and some coding guidelines.

E.4.1.1 Section 1: Define supply ports

The supply ports connected to the external supply nets shall be declared before the physical implementation. Internal supply ports, such as the output supply port of a power switch, do not need to be specified here. In this example, there are only four external supply ports—vdd1 and vdd2 for power, and vss1 and vss2 for ground—to be specified for this design.

E.4.1.2 Section 2: Define supply nets

Both internal and external supply nets need to be declared here. External supply nets are the ones connected to the external supply ports, declared in Section 1. Designers may choose to skip the declaration of external supply nets if they have the same name as the supply ports. The internal supply net is the one connected to the output of a power switch, a regulator, or any complex macro cell.

E.4.1.3 Section 3: Associate supply nets to supply sets

All supply set handles or supply sets created in previous stages shall be associated with actual supply net definitions at this stage. If a supply set is already created without the association of the supply nets for each supply function, use the **-update** option in **create_supply_set** to add the supply net information to the supply set. Otherwise, a new supply set needs to be created with the supply nets and associated with the supply set of the previously declared supply set handle using **associate_supply_set**.

In this example, none of the supply sets has been created in a previous UPF file. As a result, new supply sets are created and associated with the supply set handles. In addition, this example demonstrates what needs to be specified when the soft IP is part of a bottom-up implementation flow. In a top-down design flow, there is no need to explicitly define the supply nets at the soft IP level, and the supply nets can all be specified at the SoC level.

E.4.1.4 Section 4: Define power-switch strategy

A power switch is the physical implementation detail of a switchable power domain. Specify power switches at this stage to enable the physical implementation tool to insert the power switches.

E.4.2 UPF usage for physical implementation

As described in <u>E.3.3</u>, there are two approaches to create the UPF for physical implementation. This corresponds to UPF^3 and UPF^b in <u>Figure E.2</u>. The discussion in <u>E.3.3</u> applies to this stage as well.

E.4.3 How to use the physical implementation UPF

<u>Figure E.6</u> shows the block diagram of the example in <u>Figure E.1</u> after the physical implementation stage, with power and ground connections completed.

The key tasks of UPF driven physical implementation are the power-switch connection and the supply connection of both regular logic and the low-power logic inferred from UPF.



Figure E.6—Block example diagram after physical implementation

E.4.3.1 Power-switch connection

Even though UPF does not specify the actual power-switch network of a power domain, it has complete specification of one power switch with the logic control signal and the supply net connection specified.

Implementation tools can then use this single-switch connection as a template to create various topologies of the switch network to meet the design requirements, such as ramp-up time and rush current limit. For example, the power-switch strategy PDTop_switch in top_phy.upf specifies the output supply net of the switch is vdd_top with an input set net of vdd1 and switch enable control of sw_en1. The output supply net vdd_top is used to power all the instances in the extent of power domain PDTop.

E.4.3.2 Supply net connection

For the regular logic cells of a power domain, the power pin and ground pin of each cell are connected to the primary power net and ground net of the parent power domain, respectively. For example, the power pins of the glue logic in <u>Figure E.6</u> are connected to the power net vdd_top, which is defined as the power net of the primary supply set of power domain PDTop.

The supply net connection for special low-power logic such as isolation, level-shifter, and retention cells, needs to follow the specification in UPF.

— Isolation cell supply connection: In the configuration UPF, the option -isolation_supply_set defines the supply set for each strategy. For example, in top_soft.upf, the isolation supply set for strategy isol is SSH1. In the physical implementation UPF, the supply set handle SSH1 is associated with the supply set ss1, which is resolved with power net vdd1 and ground net vss1, as shown in top_phy.upf. Therefore, the supply nets vdd1 and vss1 are the ones to be connected to the power and ground pins of the isolation cell for isol. However, since the power domain where the isolation cell is located is a switchable domain, the cell that implements strategy isol needs to be a dual-rail isolation cell, where the primary rail of the cell is connected to the power net of PDTop (i.e., vdd_top) and the secondary rail of the cell is connected to the combo cell at the output of I1/02, which is explained as follows. If the isolation strategy has no

-isolation_supply option specified, the default isolation supply associated with the reference domain is used.

- Level-shifter supply connection: The level-shifter cell power and ground connections are determined by the options -input_supply_set and -output_supply_set in the level-shifter strategy. For example, in top_impl.upf, the level-shifter strategy lvll has an input supply set of PDTop.primary and output supply set of PDTop.SSH2. Since an enabled level-shifter is used to implement both the isolation strategy iso3 and level-shifter strategy lvll for this connection, the input power net of the cell is vdd_top and the output power net of the cell is vdd2, as shown in Figure E.6. If the two options are not specified for a level-shifter strategy, the supply set of the source is used as the input supply set and the supply set of the sink is used as the output supply set.
- State retention supply connection: In the configuration UPF, the option -retention_supply_set describes the supply set for each retention strategy. Retention cells mostly have two set of supplies. The primary rails are connected to the power and ground nets of the parent domain. The secondary rails are connected to the power and ground nets of the specified retention supply set. Even though it is not shown in Figure E.6, the retention cells that implement the strategy for ret1 in top_soft.upf shall have the following connections: primary rails of the cell are connected to vdd1_sw and vss1, respectively, and the secondary rails of the cell are connected to vdd1 and vss1, respectively.

E.5 SoC integration flow

The soft IP implemented in <u>E.4</u> needs to be integrated into SoC eventually. In a bottom-up hierarchical implementation flow, each partition/tile is implemented fully before the chip level assembly. Consequently, a SoC design looks exactly like the example shown in <u>Figure E.1</u>, and it consists of hard and soft IP blocks. As a result, the design flow described for the soft IP shown in <u>Figure E.2</u> can be applied to SoC design and implementation as well.

E.6 How to create a configuration UPF

There are two ways to create the configuration UPF, top_soft.upf, for the example in Figure E.1. One way is to code the UPF from scratch. The other approach is to start with a constraint UPF for the soft IP and then create the configuration UPF by configuring the constraint UPF under the context in which the soft IP is instantiated.

The successive refinement methodology (see <u>4.8</u>) introduced in <u>E.2</u>, <u>E.3</u>, and in <u>E.4</u> demonstrates an efficient way to create UPF models at different design stages. However, from a soft IP provider's viewpoint, even the configuration UPF may be too restrictive. For example, for a soft IP without the context in which it will be instantiated, it is very hard to come up with all the needed low-power control signals. In addition, a soft IP user may choose to configure the soft IP in a way that is completely different from another user who depends on the usage of the soft IP under different scenarios.

E.6.1 UPF constraints

Constraint UPF consists of context-independent and technology-independent power intent specifications. Constraint UPF can be delivered along with a soft IP to specify the constraints for using the soft IP in a particular context implemented with a given technology, as follows:

- *Power domains*: The only required information for power-domain constraint is the specification of the extent of the power domain by using the **create_power_domain** command.
- *Isolation values*: These are isolation value requirements for ports of the soft IP. For input ports, the constraints simply indicate that when drivers of these inputs are switched off (in the context where

the soft IP is instantiated), the specified isolation value shall be seen at those inputs. To specify the constraints, use set port attributes, as shown in top soft.upf Section 9.

- *Retention elements*: These are simply a list of registers that require state retention functionality when their parent domains are switched off. Retention constraint can be specified using set_retention_elements.
- *Power states*: Use **add_power_state** to specify the power states of the supply set handles of each domain and the power states of the system.

For example, the constraint UPF for the design top in Figure E.1 is shown as follows:

```
# Start of top level Constraint UPF, assume the file name of top constr.upf
# section 1: define power domains
create power domain PDTop -elements {.}
create power domain PD1 -elements { I3 }
# section 2: define supply set states
add power state PDTop.primary -supply\
-state {ON -simstate NORMAL -supply expr {power == FULL ON && ground ==
   FULL ON
-state {OFF -simstate CORRUPT -supply expr {power == OFF || ground == OFF}}
add power state PD1.primary -supply\
-state {ON -simstate NORMAL -supply expr {power == FULL ON && ground ==
   FULL ON}} \
-state {OFF -simstate CORRUPT -supply expr {power == OFF || ground == OFF}}
# section 3: define system level power states
add power state PDTop -domain \
-state {CS1 -logic expr {primary == ON && PD1.primary == ON}} \
-state {CS2 -logic expr {primary == ON && PD1.primary == OFF}} \
-state {CS3 -logic expr {primary == OFF && PD1.primary == OFF}}
# section 4: define isolation constraints
set port attributes -domain PDTop -applies to inputs -clamp value 0
set port attributes -domain PD1 -applies to inputs -clamp value 0
set port attributes -ports {in3} -clamp value 1
# section 5: define state retention constraints
set retention elements ret list -elements I3 -transitive TRUE
# end of Top level Constraint UPF, top constr.upf
```

The constraint UPF for the design top is quite simple. It has two power domains, PDTop and PD1, and both can be switched off according to the power state definitions. Furthermore, all domain inputs of PDTop and PD1 require an isolation value of logic 0, except for the input port in3, which requires an isolation value of logic 1. Per UPF semantics, the inputs to PDTop are the following ports: in1, in2, in3, and I2/z; and the inputs to PD1 are I3/A, I3/B, and I3/C. This example also specifies all sequential instances in I3 and its children are optional for state retention. This constraint simply implies either all or none of the sequential instances under I3 shall be state retention cells.

NOTE—The preceding commands of Section 4 also demonstrate the precedence policy described in <u>5.8</u>. The first two commands are considered a more generic description and the last three commands are considered as more specific descriptions that can overwrite the previously specified generic description for a given port.

It is clear that a constraint UPF cannot be used for simulation or implementation as it has incomplete information, e.g., the control signals for the power domains, isolation, and state retention logic are missing in the constraint UPF file. However, it contains key power intent that can be used to generate the configuration UPF when given the context of how the soft IP will be used.

E.6.2 Configure a constraint UPF into a configuration UPF

The process of generating a configuration UPF from a constraint UPF is called *configuration*. A constraint UPF shall be configured within the context of how the corresponding soft IP will be used. For the design in Figure E.1, the context of the block top is as follows:

- It has two sets of external supplies coming into the block
- The primary supplies of both domains are derived from the same supply set
- The hard IP has a UPF model

The followings are key steps to configure a constraint UPF into a configuration UPF:

a) Determine the control signals for the special low-power logic

Designers need to specify the control signals for power gating (switches), isolation logic, and state retention logic. If those control pins do not exist in RTL, designers need to create them using UPF commands, as shown in Section 2 of top_soft.upf.

b) Determine the external supply sets

In top_constr.upf, two switchable power domains are declared along with the implicit primary supply set. However, the two primary supply sets need to be switched from some external supplies that should already be known at the time of configuration, as shown in Section 3 of the configuration UPF top soft.upf.

c) Instantiate hard IP

If the soft IP contains any hard IP with UPF description, designers need to specify how the hard IP is integrated. Two key integration tasks are specifying how the hard IP supplies are connected to the supplies at the soft IP level and integrating the hard IP level system power states into the system power states at the soft IP level. Section 4 of top_soft.upf illustrates how to integrate the hard IP UPF. The power state integration is discussed in step <u>d</u>).

d) Define power states for all supply set handles

For both the newly created supply sets in step \underline{b}) and the previously declared power states in the constraint UPF, all legal power states shall be specified. For the gated supplies such as the primary supplies of the power domain PDTop and PD1, designers shall also specify from which external supplies they are gated, such as the logic expression defined in the Section 5 of the configuration UPF top_soft.upf.

e) Defining or updating the system power states

There are existing system power states defined in the constraint UPF, i.e., top_constr.upf. If the soft IP does not contain any hard IP with UPF definition, the power states can be reused for configuration UPF definition, except for the newly defined external supply set handles, which can be added using the **-update** option of **add_power_state**. However, in the example of Figure E.1, cellA is a hard IP whose UPF has also system power states defined. Therefore, in the configuration UPF, there is a need to combine the existing system power states in constraint UPF and the system power states in the hard IP UPF into the system power states for the configuration UPF. One way to perform the integration is to expand a power state in constraint UPF into multiple states for each power state in the hard IP UPF. For example, the system power state CS1 of top_constr.upf can be expanded into four states, one for each system power states that are not expected to be reached in normal operation of the soft IP.

f) Define isolation strategy

Now that the control signals for each switchable power domains are known, the designers need to create all necessary isolation strategies based on the isolation constraints defined in the constraint UPF. In top_constr.upf, the isolation constraint for both the PDTop and PD1 input ports are clamp 0 except for the primary input port in3. If designers decide not to create isolation strategies

for all primary input ports, the only needed isolation strategies to cover connections between PDTop and PD1 are iso1 and iso2 as shown in Section 7 of top soft.upf.

g) Define state retention strategy

Instead of creating the retention strategy from scratch, as the one shown in Section 8 of top_soft.upf, the following command can be used to create the retention strategy using the retention list created in top constr.upf.

set_retention ret1 -domain PD1 -elements {ret_list}
-retention_supply_set PDTop.SSH1 \
-save_signal {ret_en negedge } -restore_signal {ret_en posedge}

The configuration UPF created from updating the constraint UPF is as follows:

```
# A configuration UPF configured from the constraint UPF, assume the file name
   of top soft2.upf
source top constr.upf
source cellA.upf
create_logic_port sw_en1 -direction in
create_logic_port sw_en2 -direction in
create logic port iso en1 -direction in
create logic port iso en2 -direction in
create_logic_port retention -direction in
create supply set ss1
create supply set ss2
create power domain PDTop -supply SSH1 -supply SSH2 -update
create power switch PDTop sw -output supply port {out PDTop.primary.power} \
-input supply port {in PDTop.SSH1.power} -control port {ctrl sw en1} \
-on state { ON in !ctrl }
create power switch PD1 sw -output supply port {out PD1.primary.power} \
   -input supply port {in PDTop.SSH1.power} -control port {ctrl sw en2}\
-on state { ON in !ctrl }
apply power model upf modelA -scope I2 -supply map { { PD.SSAH PDTop.SSH1 }
   {PD.SSBH PDTop.SSH2} }
add power state PDTop.SSH1 \
-state {ON -simstate NORMAL -supply expr {power == FULL ON && ground ==
   FULL ON}}
-state {OFF -simstate CORRUPT -supply expr {power == OFF || ground == OFF}}
add_power_state PDTop.SSH2 \
-state {ON -simstate NORMAL -supply expr {power == FULL ON && ground ==
   FULL ON}}\
-state {OFF -simstate CORRUPT -supply expr {power == OFF || ground == OFF}}
add power state PDTop \setminus
-state {S1 -logic expr \
{ SSH1 == ON && SSH2 == ON && primary == ON && PD1.primary == ON && I2/PD ==
   S1}} \
-state {S2 -logic expr \
{ SSH1 == ON && SSH2 == ON && primary == ON && PD1.primary == OFF && I2/PD ==
   S1}} \
-state {S3 -logic expr \
{ SSH1 == ON && SSH2 == ON && primary == OFF && PD1.primary == OFF && I2/PD ==
   S1\} \} \setminus
-state {S4 -logic expr \
{ SSH1 == ON && SSH2 == ON && primary == ON && PD1.primary == ON && I2/PD ==
   S2}} \
-state {S5 -logic expr \
{ SSH1 == ON && SSH2 == ON && primary == ON && PD1.primary == OFF && I2/PD ==
   S2}} \
```

```
-state {S6 -logic expr \
{ SSH1 == ON && SSH2 == ON && primary == OFF && PD1.primary == OFF && I2/PD ==
   S2}} \
-state {S7 -logic expr \
{ SSH1 == ON && SSH2 == OFF && primary == OFF && PD1.primary == OFF && I2/PD
   == S3}} \
-state {S8 -logic expr \
{ SSH1 == ON && SSH2 == OFF && primary == OFF && PD1.primary == OFF && I2/PD
   == S4}} \
-state {S9 -logic expr \
{ SSH1 == OFF && SSH2 == OFF && primary == OFF && PD1.primary == OFF && I2/PD
   == S4}}
set isolation isol -domain PDTop -applies to output \
-isolation supply set PDTop.SSH1 -location self \
-isolation signal iso en1 -isolation sense high -clamp value 0 -
   diff supply only TRUE
set isolation iso2 -domain PD1 -source PD1.primary \
-isolation supply set PDTop.SSH1 -location parent \setminus
-isolation_signal iso_en2 -isolation_sense high -clamp value 0 -
   diff supply only TRUE
set_isolation iso3 -domain PDTop -source PDTop.primary -sink PDTop.SSH1 \
-isolation_supply_set PDTop.SSH1 -location self \
-isolation_signal iso_en1 -isolation_sense high -clamp_value 1 -
   diff supply only TRUE
set_isolation iso4 -domain PDTop -source I2/SSBH_SW \
-isolation supply set PDTop.SSH2 -location parent \
-isolation signal in2 -isolation sense high -clamp value 1 -diff supply only
   TRUE
set retention ret1 -domain PD1 -retention supply set PDTop.SSH1 -elements
   ret list \setminus
-save signal {ret en negedge} -restore signal {ret en posedge}
# end of top soft2.upf
```

Annex F

(normative)

Value conversion tables

The predefined value conversion tables (VCTs) are as follows.

F.1 VHDL_SL2UPF

```
create_hdl2upf_vct VHDL_SL2UPF
-hdl_type vhdl
-table { {'U' UNDETERMINED}
      {'X' UNDETERMINED}
      {'O' OFF}
      {'1' FULL_ON}
      {'Z' UNDETERMINED}
      {'L' OFF}
      {'H' FULL_ON}
      {'W' UNDETERMINED}
      {'-' UNDETERMINED} }
```

F.2 UPF2VHDL_SL

F.3 VHDL_SL2UPF_GNDZERO

```
create_hdl2upf_vct VHDL_SL2UPF_GNDZERO
-hdl_type vhdl
-table { {'U' UNDETERMINED}
        {'U' UNDETERMINED}
        {'0' FULL_ON}
        {'1' OFF}
        {'Z' UNDETERMINED}
        {'L' FULL_ON}
        {'H' OFF}
        {'W' UNDETERMINED}
        {'-' UNDETERMINED}}
```

F.4 UPF_GNDZERO2VHDL_SL

```
create_upf2hdl_vct UPF_GNDZERO2VHDL_SL
-hdl_type vhdl
-table {{UNDETERMINED 'X'}
        {PARTIAL_ON 'X'}
        {OFF '1'}
        {FULL_ON '0'}}
```

F.5 SV_LOGIC2UPF

```
create_hdl2upf_vct SV_LOGIC2UPF
-hdl_type sv
-table {{X UNDETERMINED}
        {Z UNDETERMINED}
        {1 FULL_ON }
        {0 OFF }}
```

F.6 UPF2SV_LOGIC

F.7 SV_LOGIC2UPF_GNDZERO

F.8 UPF_GNDZERO2SV_LOGIC

```
create_upf2hdl_vct UPF_GNDZER02SV_LOGIC
-hdl_type sv
-table {{UNDETERMINED X}
        {PARTIAL_ON X}
        {OFF 1}
        {FULL_ON 0}}
```

F.9 VHDL_TIED_HI

F.10 SV_TIED_HI

F.11 VHDL_TIED_LO

F.12 SV_TIED_LO

Annex G

(normative)

Supporting hard IP

When a block has an input port and an output port that are directly connected internally by a logical net, the two ports involved are called *feedthrough ports*. Tools need to recognize such ports in order to traverse through them to identify the true source(s) and sink(s) of a net. For a hard IP, automatically recognizing such ports may be difficult. To explicitly identify feedthrough ports of a hard IP, use the **feedthrough** option in a **set_port_attributes** command (see <u>6.46</u>).

When a hard IP has input ports and/or output ports that are not connected internally, such ports need not be considered for any power intent specification. In addition, when performing analysis on the need of isolation or level-shifter logic at the interface of the hard IP, these ports shall be ignored. To model such ports, use the **unconnected** option in a **set_port_attributes** command (see <u>6.46</u>).

G.1 Attributing feedthrough ports of hard IP

In this case, the port list shall specify the ports of a model that are all connected electrically by the same metal wire. If the specified model has a functional (i.e., behavioral simulation model) or physical (i.e., layout) description, it is an error if the specified ports are not directly connected to each other in the functional or physical model description. If the specified ports are not defined in the corresponding model description, the attributes are ignored.

Tools shall be able to traverse through the connected ports when performing driver and load analysis in the scope where the model is instantiated.

Example

Assume a macro cell with the following internal structure (see Figure G.1), where the cell has:

- two set of supplies: vddA / vssA and vddB / vssB
- Il drives logic powered by vddA / vssA
- I2 has direct connection to port O1 and O2
- I3 drives logic powered by vddB / vssB and connection to port O3
- I4 does not drive any logic internally
- 04 is driven by logic powered by vddB / vssB

The following commands described the internal connection for input ports I2 and I3, and output ports O1, O2, and O3 of the cell:

set_port_attributes -ports {I2 01 02} -model cellX -feedthrough
set port attributes -ports {I3 03} -model cellX -feedthrough

NOTE—Since input port I3 also drives internal logic, it is allowed to have a **-receiver_supply** attribute set on port I3 as well when a UPF power model is created for this cell.



Figure G.1—Hard IP macro cell

In the following example,

set_port_attributes -ports {I2 01} -model cellX -feedthrough
set port attributes -ports {I2 02} -model cellX -feedthrough

the first command connects 12 to 01, the second command connects 02 to 12. As a result, 12, 01, and 02 are all connected together, which is equivalent to the following:

set port attributes -ports {I2 O1 O2} -model cellX -feedthrough

Another way to specify the attribute is to use the corresponding UPF port attribute UPF_feedthrough directly, see 5.6.

G.2 Attributing unconnected ports of hard IP

In this case, the **set_port_attributes** command, with the **-unconnected** option (see <u>6.46</u>), specifies a list of ports of a model that are not connected to any internal logic. If such a port is an input port, it means there is no logic within the model driven by the port; if such a port is an output port, it means there is no logic within the model driving the port. These ports shall not be associated with any other port attributes. This attribute also overwrites any default supply net or supply set association with respect to the specified ports, i.e., the specified ports are not associated with any supply net or supply set in UPF.

If the specified model has a functional (i.e., behavioral simulation model) or physical (i.e., layout) description, it is an error if the specified ports are connected to any logic or are part of the function definition of the functional or physical model description. If the specified ports are not defined in the corresponding model description, the attributes are ignored.

For simulation semantics, tools shall consider the signal driven by the specified ports as corrupted.

Another way to specify the attribute is to use the corresponding UPF port attribute **UPF_unconnected** directly, see 5.6.

Example

In the example from $\underline{G.1}$, the input port I4 is not connected to any internal logic. The following commands can be used to attribute that port as an unconnected one:

set port attributes -ports {I4} -model cellX -unconnected

Annex H

(normative)

UPF power-management commands semantics and Liberty mappings

H.1 Introduction

This annex describes how the information specified in each power-management cell command (see <u>Clause 7</u>) can be used by the corresponding power intent commands in <u>Clause 6</u>. In addition, it also describes the mapping between each command and option to the Liberty attributes. Unless otherwise stated, the referenced Liberty attributes are based on the Liberty 2009.06 release (see [B7]).

For designers who prefer to use the Liberty approach to describe power-management cell attributes, the mapping tables in this annex can be used to understand what the required information is in Liberty to enable a UPF flow.

H.1.1 Liberty attribute mapping

If a UPF option has a corresponding Liberty attribute, the following type of mapping table (see <u>Table H.1</u>) is used:

Name	Group	Туре	Value
pg_type	pg_pin	string	primary_ground

Table H.1—Sample Liberty attribute mapping

where the column *Name* lists the corresponding Liberty attribute name; the column *Group* indicates the name of the group statement in which this attribute is specified; the column *Type* indicates the attribute type such as a string, Boolean, integer, or floating point; and the column *Value* indicates the corresponding attribute value.

If a UPF option has no corresponding Liberty attribute, this will be indicated explicitly.

H.1.2 Potential conflicts with library command definitions

These mappings are based on the syntax from the actual library command definitions (see <u>Clause 7</u>), which are replicated in this annex as a convenience. In the event of a conflict between this material and the syntax shown in <u>Clause 7</u>, the syntax listing for <u>Clause 7</u> shall prevail.

H.2 define_always_on_cell

```
define_always_on_cell [from 7.2]
-cells cell_list
-power pin
```

-ground pin [-power_switchable pin] [-ground_switchable pin] [-isolated_pins list_of_pin_lists][-enable expression_list]

The Liberty mappings for this command are as follows:

a) <u>Table H.2</u> indicates the Liberty attribute mapping for all cells identified by the **-cells** option of this command.

Table H.2—Liberty attribute mapping for -cells

Name	Group	Туре	Value
always_on	cell	Boolean	true

b) <u>Table H.3</u> indicates the Liberty attribute mapping for the **-power** argument.

Table H.3—Liberty attribute mapping for -power

Name	Group	Туре	Value
pg_type	pg_pin	string	backup_power primary_power

- 1) If this option is specified with **-power_switchable**, the corresponding *pg_type* is **backup_power**. During implementation, this pin is connected to the ground net specified by users.
- 2) If this option is not specified with **-power_switchable**, the corresponding *pg_type* is **primary_power**. During implementation, this pin is connected to the ground net of the primary supply set of the power domain in which the cell is located.
- c) <u>Table H.4</u> indicates the Liberty attribute mapping for the **-ground** argument.

Table H.4—Liberty attribute mapping for -ground

Name	Group	Туре	Value
pg_type	pg_pin	string	backup_ground primary_ground

- 1) If this option is specified with **-ground_switchable**, the corresponding *pg_type* is **backup_ground**. During implementation, this pin is connected to the ground net specified by users.
- 2) If this option is not specified with -ground_switchable, the corresponding pg_type is primary_ground. During implementation, this pin is connected to the ground net of the primary supply set of the power domain in which the cell is located.

d) <u>Table H.5</u> indicates the Liberty attribute mapping for the **-power_switchable** argument.

Name	Group	Туре	Value
pg_type	pg_pin	string	primary_power

Table H.5—Liberty attribute mapping for -power_switchable

During implementation, this pin is connected to the power net of the primary supply set of the power domain in which the cell is located.

e) <u>Table H.6</u> indicates the Liberty attribute mapping for the **-ground_switchable** argument.

Table H.6—Liberty attribute mapping for -ground_switchable

Name	Group	Туре	Value
pg_type	pg_pin	string	primary_ground

During implementation, this pin is connected to the ground net of the primary supply set of the power domain in which the cell is located.

- f) -isolated_pins has no corresponding Liberty attribute.
- g) -enable has no corresponding Liberty attribute.

H.3 define_diode_clamp

```
define_diode_clamp [from 7.3]
    -cells cell_list
    -data_pins pin_list
    [-type <power | ground | both>]
    [-power pin] [-ground pin]
```

The Liberty mappings for this command are as follows:

a) <u>Table H.7</u> indicates the Liberty attribute mapping for all cells identified by the **-cells** option of this command.

Name	Group	Туре	Value
antenna_diode_type	cell	Boolean	true

b) -data_pins has no corresponding Liberty attribute.

c) -type has no corresponding Liberty attribute.

- d) <u>Table H.8</u> indicates the Liberty attribute mapping for the **-power** argument.
- e) <u>Table H.9</u> indicates the Liberty attribute mapping for the **-ground** argument.

Table H.8—Liberty attribute mapping for -power

Name	Group	Туре	Value
pg_type	pg_pin	string	primary_power

Table H.9—Liberty attribute mapping for -ground

Name	Group	Туре	Value
pg_type	pg_pin	string	primary_ground

H.4 define_isolation_cell

```
define_isolation_cell [from 7.4]
    -cells cell_list
    [-power power_pin]
    [-ground power_pin]
    {-enable pin [-clamp_cell < high | low>]
    | -pin_groups {{input_pin output_pin [enable_pin]}*}
    | -no_enable < high | low | hold>}
    [-always_on_pins pin_list]
    [-aux_enables ordered_pin_list]
    [-power_switchable power_pin] [-ground_switchable ground_pin]
    [-valid_location <source | sink | on | off | any>]
    [-no_dedicated]
```

The Liberty mappings for this command are as follows:

a) <u>Table H.10</u> indicates the Liberty attribute mapping for all cells identified by the **-cells** option of this command.

Table H.10—Liberty attribute mapping for -cells

Name	Group	Туре	Value
is_isolation_cell	cell	Boolean	true

b) <u>Table H.11</u> and <u>Table H.12</u> indicate the Liberty attribute mapping for the **-power** argument.

Table H.11—Liberty attribute mapping for -power and -power_switchable

Name	Group	Туре	Value
pg_type	pg_pin	string	backup_power

 This mapping takes place when the cell is also specified with the -power_switchable option. In this case, tools shall connect the pin to the power net of the isolation supply set in the corresponding isolation strategy or the power net of the default isolation supply set of the power domain in the corresponding isolation strategy unless the connection is specified explicitly.

Table H.12—Liberty attribute mapping for -power

Name	Group	Туре	Value
pg_type	pg_pin	string	primary_power

- 2) This mapping takes place when the cell is not specified with the **-power_switchable** option. In this case, tools shall connect the pin to power net of the primary supply set of the power domain in which the cell is located.
- c) <u>Table H.13</u> and <u>Table H.14</u> indicate the Liberty attribute mapping for the **-ground** argument.

Table H.13—Liberty attribute mapping for -ground and -ground_switchable

Name	Group	Туре	Value
pg_type	pg_pin	string	backup_ground

 This mapping takes place when the cell is also specified with the -ground_switchable option. In this case, tools shall connect the pin to the ground net of the isolation supply set in the corresponding isolation strategy or the ground net of the default isolation supply set of the power domain in the corresponding isolation strategy unless the connection is specified explicitly.

Table H.14—Liberty attribute mapping for -ground

Name	Group	Туре	Value
pg_type	pg_pin	string	primary_ground

- 2) This mapping takes place when the cell is not specified with the **-ground_switchable** option. In this case, tools shall connect the pin to ground net of the primary supply set of the power domain in which the cell is located.
- d) <u>Table H.15</u> indicates the Liberty attribute mapping for the **-enable** argument.

Table H.15—Liberty attribute mapping for -enable

Name	Group	Туре	Value
isolation_cell_enable_pin	pin	Boolean	true

Tools need to connect the enable pin to the isolation signal specified in the corresponding isolation strategy.

- e) -clamp_cell has no corresponding Liberty attribute.
 - 1) For a clamp high cell, tools can presume the following connections unless they are specified explicitly:
 - i) Connect the data pin to the net or pin targeted for isolation;
 - ii) Connect the enable pin to the isolation signal specified in the corresponding isolation strategy;
 - iii) Connect the power pin of the cell to the power net of the isolation supply set in the corresponding isolation strategy or the power net of the default isolation supply set of the power domain in the corresponding isolation strategy.
 - 2) For a clamp low cell, tools can presume the following connections unless they are specified explicitly:
 - i) Connect the data pin to the net or pin targeted for isolation;
 - ii) Connect the enable pin to the isolation signal specified in the corresponding isolation strategy;
 - iii) Connect the ground pin of the cell to the ground net of the isolation supply set in the corresponding isolation strategy or the ground net of the default isolation supply set of the power domain in the corresponding isolation strategy.
- f) For **-pin_groups**, the corresponding modeling of a multi-bit isolation cell is the bundle group in Liberty. Within the bundle group, standard pin attributes can be used for the isolation data pin and enable pin.
- g) -no_enable has no corresponding Liberty attribute.
- h) <u>Table H.16</u> indicates the Liberty attribute mapping for the **-always_on_pins** argument.

Table H.16—Liberty	/ attribute	mapping	for	-always_	_on_	_pins
--------------------	-------------	---------	-----	----------	------	-------

Name	Group	Туре	Value
always_on	pin	Boolean	true

i) **-aux_enables** has no corresponding Liberty attribute.

This option models isolation cells with more than one enable pins. The index 0 is reserved for the isolation enable pin specified by the **-enable** option. The pins listed in this option start with index 1. To use such cells for isolation, the corresponding strategy needs to be specified with a signal list in the **-isolation_signal** option. The elements in the list are ordered with the index starting with 0. The signals in the list should be connected to the pins of the cells with the same index.

j) <u>Table H.17</u> indicates the Liberty attribute mapping for the **-power_switchable** argument.

Table H.17—Liberty attribute mapping for -power_switchable

Name	Group	Туре	Value
pg_type	pg_pin	string	primary_power

Tools need to connect the pin to the power net of the primary supply set of the power domain in which the cell is located.

k) <u>Table H.18</u> indicates the Liberty attribute mapping for the **-ground_switchable** argument.

Name	Group	Туре	Value
pg_type	pg_pin	string	primary_ground

Table H.18—Liberty attribute mapping for -ground_switchable

Tools need to connect the pin to the ground net of the primary supply set of the power domain in which the cell is located.

1) -valid_location has no corresponding Liberty attribute.

Verification tools need to ensure the implementation of the isolation strategy places the isolation cells in the correct location based on this definition.

m) -non_dedicated has no corresponding Liberty attribute.

H.5 define_level_shifter_cell

define_level_shifter_cell [from 7.5]
 -cells cell_list
 [-input_voltage_range {voltage_ranges}] [-output_voltage_range {voltage_ranges}]
 [-ground_input_voltage_range {voltage_ranges}]
 [-ground_output_voltage_range {voltage_ranges}]
 [-direction <low_to_high | high_to_low | both>]
 [-input_power_pin power_pin]
 [-output_power_pin power_pin]
 [-output_ground_pin ground_pin]
 [-output_ground_pin ground_pin]
 [-ground ground_pin] [-power power_pin]
 [-enable pin | -pin_groups {{input_pin output_pin [enable_pin]}*}]
 [-valid_location <source | sink | either | any>]
 [-bypass_enable expression] [-multi_stage integer]

The Liberty mappings for this command are as follows:

a) <u>Table H.19</u> indicates the Liberty attribute mapping for all cells identified by the **-cells** option of this command.

Name	Group	Туре	Value
is level shifter	cell	Boolean	true

Table H.19—Liberty attribute mapping for -cells

b) -input_voltage_range has no corresponding Liberty attribute.

The syntax of this attribute is different from the Liberty attribute input_voltage_range, which specifies only two values to indicate the voltage lower bound and upper bound.

c) **-output_voltage_range** has no corresponding Liberty attribute.

The syntax of this attribute is different from the Liberty attribute output_voltage_range, which specifies only two values to indicate the voltage lower bound and upper bound.

d) **-ground_input_voltage_range** has no corresponding Liberty attribute.

The syntax of this attribute is different from the Liberty attribute input_voltage_range, which specifies only two values to indicate the voltage lower bound and upper bound.

e) -ground_output_voltage_range has no corresponding Liberty attribute.

The syntax of this attribute is different from the Liberty attribute output_voltage_range, which specifies only two values to indicate the voltage lower bound and upper bound.

- f) -direction has no corresponding Liberty attribute.
- g) <u>Table H.20</u> indicates the Liberty attribute mapping for the -input_power_pin argument.

Table H.20—Liberty attribute mapping for -input_power_pin

Name	Group	Туре	Value
pg_type	pg_pin	string	primary_power

Tools need to connect the pin to the power net of the input supply set in the corresponding levelshifter strategy [identified by the **-input_supply_set** of **set_level_shifter** (see <u>6.43</u>)] or the power net of the driving cell of the level-shifter, unless the connection is specified explicitly.

h) <u>Table H.21</u> indicates the Liberty attribute mapping for the **-output_power_pin** argument.

Table H.21—Liberty attribute mapping for -output_power_pin

Name	Group	Туре	Value
pg_type	pg_pin	string	primary_power

Tools need to connect the pin to the power net of the output supply set in the corresponding levelshifter strategy [identified by the **-output_supply_set** of **set_level_shifter** (see <u>6.43</u>)] or the power net of the load cell of the level-shifter, unless the connection is specified explicitly.

i) <u>Table H.22</u> indicates the Liberty attribute mapping for the **-input_ground_pin** argument.

Table H.22—Liberty attribute mapping for -input_ground_pin

Name	Group	Туре	Value
pg_type	pg_pin	string	primary_ground

Tools need to connect the pin to the ground net of the input supply set in the corresponding levelshifter strategy [identified by the -input_supply_set of set_level_shifter (see 6.43)] or the ground net of the driving cell of the level-shifter, unless the connection is specified explicitly.

j) <u>Table H.23</u> indicates the Liberty attribute mapping for the **-output_ground_pin** argument.

Tools need to connect the pin to the ground net of the output supply set in the corresponding levelshifter strategy [identified by the **-output_supply_set** of **set_level_shifter** (see <u>6.43</u>)] or the ground net of the load cell of the level-shifter, unless the connection is specified explicitly.

Table H.23—Liberty attribute mapping for -output_ground_pin

Name	Group	Туре	Value
pg_type	pg_pin	string	primary_ground

k) <u>Table H.24</u> indicates the Liberty attribute mapping for the **-ground** argument.

|--|

Name	Group	Туре	Value
pg_type	pg_pin	string	primary_ground

Tools need to connect the pin to ground net of the primary supply set of the power domain in which the cell is located.

1) <u>Table H.25</u> indicates the Liberty attribute mapping for the **-power** argument.

Table H.25—Liberty attribute mapping for -power

Name	Group	Туре	Value
pg_type	pg_pin	string	primary_power

Tools need to connect the pin to power net of the primary supply set of the power domain in which the cell is located.

m) <u>Table H.26</u> indicates the Liberty attribute mapping for the **-enable** argument.

Table H.26—Liberty	/ attribute	mapping for	or -enable
--------------------	-------------	-------------	------------

Name	Group	Туре	Value
level_shifter_enable_pin	pin	Boolean	true

- n) For **-pin_groups**, the corresponding modeling of a multi-bit isolation cell is the bundle group in Liberty. Within the bundle group, standard pin attributes can be used for the isolation data pin and enable pin.
- o) **-valid_location** has no corresponding Liberty attribute.

Verification tools need to ensure the implementation of the level-shifter strategy places the levelshifter in the correct location based on this definition.

p) **-bypass_enable** has no corresponding Liberty attribute.

The polarity of the bypass enable pin can be derived from the Liberty attribute level_shifter_data_pin and the function of the output pin.

q) **-multi_stage** has no corresponding Liberty attribute.

H.6 define_power_switch_cell

define_power_switch_cell [from 7.6]
 -cells cell_list
 -type <footer | header>
 -stage_1_enable expression [-stage_1_output expression]
 {-power_switchable power_pin -power power_pin
 | -ground_switchable ground_pin -ground ground_pin]}
 [-stage_2_enable expression [-stage_2_output expression]]
 [-always_on_pins ordered_pin_list]
 [-gate_bias_pin power_pin]

The Liberty mappings for this command are as follows:

a) <u>Table H.27</u> indicates the Liberty attribute mapping for all cells identified by the **-cells** option of this command.

Table H.27—Liberty attribute mapping for -cells

Name	Group	Туре	Value
switch_cell_type	cell	Boolean	coarse_grain

- b) For **-type**, if a cell has a pg_pin with pg_type internal_power in the Liberty definition, then the cell is a header cell; if a cell has a pg_pin with pg_type internal_ground, then the cell is a footer cell.
- c) -stage_1_enable (-stage_2_enable) has no corresponding Liberty attribute(s).
 - 1) The Liberty pin attribute does not differentiate the function between the two enables, so two user attributes are created here. However, the Liberty pin attribute switch_function can be used to describe the switch function on the switched pg_pin, which has pg_type of either internal power or internal ground.
 - 2) Tools need to connect the pins to the switch-enable signal specified in the **-control_port** option of the corresponding **create_power_switch** command (see <u>6.18</u>).
- d) <u>Table H.28</u> indicates the Liberty attribute mapping for the **-power_switchable** argument.

Table H.28—Liberty attribute mapping for -power_switchable

Name	Group	Туре	Value
pg_type	pg_pin	string	internal_power

Tools need to connect the pin to the supply net specified by the **-output_supply_port** option of the corresponding **create_power_switch** (see 6.18) or **set_power_switch** (see 6.47) commands.

e) <u>Table H.29</u> indicates the Liberty attribute mapping for the **-power** argument.

Table H.29—Liberty attribute mapping for -power

Name	Group	Туре	Value
pg_type	pg_pin	string	primary_power

Tools need to connect the pin to the supply net specified by the **-input_supply_port** option of the corresponding **create_power_switch** (see 6.18) or **set_power_switch** (see 6.47) commands.

f) <u>Table H.30</u> indicates the Liberty attribute mapping for the **-ground_switchable** argument.

Table H.30—Liberty attribute mapping for -ground_switchable

Name	Group	Туре	Value
pg_type	pg_pin	string	internal_ground

Tools need to connect the pin to the supply net specified by the **-output_supply_port** option of the corresponding **create_power_switch** (see 6.18) or **set_power_switch** (see 6.47) commands.

g) <u>Table H.31</u> indicates the Liberty attribute mapping for the **-ground** argument.

Table H.31—Liberty attribute mapping for -ground

Name	Group	Туре	Value
pg_type	pg_pin	string	primary_ground

Tools need to connect the pin to the supply net specified by the **-input_supply_port** option of the corresponding **create_power_switch** (see 6.18) or **set_power_switch** (see 6.47) commands.

h) For -stage_1_output (-stage_2_output), the corresponding output pin can be automatically identified, based on the pin function and the stage_1_enable and stage_2_enable attributes.

Tools need to connect the pins to the switch-enable signal specified in the **-ack_port** option of the corresponding **create_power_switch** command (see 6.18).

i) <u>Table H.32</u> indicates the Liberty attribute mapping for the **-always_on_pins** argument.

Table H.32—Liberty attribute mapping for -always_on_pins

Name	Group	Туре	Value
always_on	pin	Boolean	true

j) <u>Table H.33</u> indicates the Liberty attribute mapping for the **-gate_bias_pin** argument.

Table H.33—Liberty attribute mapping for -gate_bias_pin

Name	Group	Туре	Value
user_pg_type	pg_pin	string	gate_bias

H.7 define_retention_cell

define_retention_cell [from 7.7]
 -cells cell_list
 -power power_pin
 -ground ground_pin
 [-cell_type string]
 [-always_on_pins pin_list]
 [-restore_function {{pin < high | low | posedge | negedge}}]
 [-save_function {{pin < high | low | posedge | negedge}}]
 [-restore_check expression] [-save_check expression]
 [-retention_check expression] [-hold_check pin_list]
 [-always_on_components component_list]
 [-power_switchable power_pin] [-ground_switchable ground_pin]</pre>

The Liberty mappings for this command are as follows:

a) <u>Table H.34</u> indicates the Liberty attribute mapping for all cells identified by the **-cells** option of this command.

Table H.34—Libe	rtv attribute	mapping	for -cells

Name	Group	Туре	Value
retention_cell	cell	string	cell_type

The cell type is the same string specified in the option -cell_type (see <u>Table H.39</u>).

b) <u>Table H.35</u> and <u>Table H.36</u> indicate the Liberty attribute mapping for the **-power** argument.

Table H.35—Liberty attribute mapping for -power and -power_switchable

Name	Group	Туре	Value
pg_type	pg_pin	string	backup_power

 This mapping takes place when the cell is also specified with the -power_switchable option. In this case, tools shall connect the pin to the power net of the retention supply set in the corresponding retention strategy or the power net of the default retention supply set of the power domain in the corresponding retention strategy unless the connection is specified explicitly.

Table H.36—Liberty attribute mapping for -power

Name	Group	Туре	Value
pg_type	pg_pin	string	primary_power

2) This mapping takes place when the cell is not specified with the **-power_switchable** option. In this case, tools shall connect the pin to power net of the primary supply set of the power domain in which the cell is located.

c) <u>Table H.37</u> and <u>Table H.38</u> indicate the Liberty attribute mapping for the **-ground** argument.

Table H.37—Liberty attribute mapping for -ground and -ground_switchable

Name	Group	Туре	Value
pg_type	pg_pin	string	backup_ground

1) This mapping takes place when the cell is also specified with the **-ground_switchable** option. In this case, tools shall connect the pin to the ground net of the retention supply set in the corresponding retention strategy or the ground net of the default retention supply set of the power domain in the corresponding retention strategy unless the connection is specified explicitly.

Table H.38—Liberty attribute mapping for -ground

Name	Group	Туре	Value
pg_type	pg_pin	string	primary_ground

- 2) This mapping takes place when the cell is not specified with the -ground_switchable option. In this case, tools shall connect the pin to ground net of the primary supply set of the power domain in which the cell is located.
- d) <u>Table H.39</u> indicates the Liberty attribute mapping for the **-cell_type** argument.

Table H.39—Liberty attribute mapping for -cell_type

Name	Group	Туре	Value
retention_cell	cell	string	user_string

e) <u>Table H.40</u> indicates the Liberty attribute mapping for the **-always_on_pins** argument.

Table H.40—Liberty attribute mapping for -always_on_pins

Name	Group	Туре	Value
always_on	pin	Boolean	true

f) <u>Table H.41</u> indicates the Liberty attribute mapping for the **-restore_function** argument.

Table H.41—Libert	y attribute mapping	for -restore	_function
-------------------	---------------------	--------------	-----------

Name	Group	Туре	Value
retention_pin	pin	string	restore save_restore

- 1) The pin shall be specified by the retention_pin attribute in Liberty. If the cell has only one retention pin, then the corresponding attribute value is save_restore; otherwise the corresponding value is restore.
- 2) <u>Table H.42</u> indicates the Liberty attribute mapping for the retention control pin functionality.

Table H.42—Liberty attribute mapping for -retention_action

Name	Group	Туре	Value
restore_action	pin	complex	<l f="" h="" r="" =""></l>

- i) The pin shall also be specified by the retention pin attribute in Liberty.
- ii) The mapping of the Liberty value to the UPF value is:

L: low

H: high

R: posedge

F: negedge

- iii) Tools need to connect the pin to the signal specified in the -restore_signal option of the set_retention command (see <u>6.49</u>). The polarity or edge-sensitivity specification of the two options shall be identical.
- g) <u>Table H.43</u> indicates the Liberty attribute mapping for the **-save_function** argument.

Table H.43—Liberty attribute mapping for -save_function

Name	Group	Туре	Value
retention_pin	pin	string	save save_restore

- 1) The pin shall be specified by the retention_pin attribute in Liberty. If the cell has only one retention pin, then the corresponding attribute value is save_restore; otherwise the corresponding value is save.
- 2) <u>Table H.44</u> indicates the Liberty attribute mapping for the retention control pin functionality.

Table H.44—Liberty attribute mapping for -retention_action

Name	Group	Туре	Value
save_action	pin	complex	<l f="" h="" r="" =""></l>

- i) The pin shall also be specified by the retention pin attribute in Liberty.
- ii) The mapping of the Liberty value to the UPF value is:
 - L: low
 - H: high
 - R: posedge
 - F: negedge
- iii) Tools need to connect the pin to the signal specified in the -save_signal option of the set_retention command (see <u>6.49</u>). The polarity or edge-sensitivity specification of the two options shall be identical.
- h) -restore_check has no corresponding Liberty attribute.
- i) -save_check has no corresponding Liberty attribute.
- j) -retention_check has no corresponding Liberty attribute.
- k) -hold_check has no corresponding Liberty attribute.
- 1) -always_on_components has no corresponding Liberty attribute.
- m) <u>Table H.45</u> indicates the Liberty attribute mapping for the **-power_switchable** argument.

Table H.45—Liberty attribute mapping for -power_switchable

Name	Group	Туре	Value
pg_type	pg_pin	string	primary_power

Tools need to connect the pin to power net of the primary supply set of the power domain in which the cell is located.

n) <u>Table H.46</u> indicates the Liberty attribute mapping for the **-ground_switchable** argument.

Table H.46—Liberty attribute mapping for -ground_switchable

Name	Group	Туре	Value
pg_type	pg_pin	string	primary_ground

Tools need to connect the pin to the ground net of the primary supply set of the power domain in which the cell is located.
Annex I

(informative)

Power-management cell modeling examples

This annex show how to model the power-management cells defined in <u>Clause 7</u>.

I.1 Modeling always-on cells

This subclause shows examples for how to model various types of always-on cells.

I.1.1 Types of always-on cells

An *always-on cell* is simply a library cell with more than one set of power and ground pins that can remain functional even when the supply to the rail-connected power or ground pin is switched off, as long as the non-switchable power or ground remains on. An always-on cell shall have at least a non-switchable power or a non-switchable ground pin defined.

A cell called always-on does not mean the cell can never be powered off. When the supply to the nonswitchable power or ground of such cell is switched off, the cell becomes non-functional. In other words, the term *always-on* actually means relative always-on.

Any logic function can be implemented in the form of an always-on cell, such as an always-on buffer, always-on inverter, always-on AND gate, or even always-on flop. In the following subclauses, several different types of always-on cells are used as examples to describe how to use the **define_always_on_cell** command (see 7.2).

- Modeling a power-switched always-on buffer
- Modeling a ground-switched always-on buffer
- Modeling a power- and ground-switched always-on buffer
- Modeling a power-switched always-on flop with internal isolation

I.1.2 Modeling a power-switched always-on buffer

To model a power-switched always-on buffer, use the **define_always_on_cell** command (see <u>7.2</u>) with the following options:

```
define_always_on_cell
-cells cells
-power pin -power_switchable pin -ground pin
```

In <u>Figure I.1</u>, a type of power-switched always-on buffer is shown. The cell's rail connection VSW is not used by the cell. The actual power of the cell comes from VDD, which needs to be routed separately. The following command models this type of cells:

```
define_always_on_cell
   -cells LP_Buf_Pow
   -power VDD -power_switchable VSW -ground VSS
```

The same command can also be used to describe any other type of power-switched always-on cells, such as an inverter, AND gate, etc.



Figure I.1—Power-switched always-on buffer

I.1.3 Modeling a ground-switched always-on buffer

To model a ground-switched always-on buffer, use the **define_always_on_cell** command (see <u>7.2</u>) with the following options:

```
define_always_on_cell
    -cells cells
    -power pin -ground_switchable pin -ground pin
```

In <u>Figure I.2</u>, a type of ground-switched always-on buffer is shown. The cell's rail connection GSW is not used by the cell. The actual ground of the cell comes from VSS, which needs to be routed separately. The following command models this type of cells:

```
define_always_on_cell
   -cells LP_Buf_Gnd
   -ground VSS -power VDD -ground_switchable GSW
```

The same command can also be used to describe any other type of ground-switched always-on cells, such as an inverter, AND gate, etc.





Figure I.2—Ground-switched always-on buffer

I.1.4 Modeling a power- and ground-switched always-on buffer

To model a power- and ground-switched always-on buffer, use the **define_always_on_cell** command (see <u>7.2</u>) with the following options:

define_always_on_cell -cells cells -power_switchable pin -ground_switchable pin -power pin -ground pin

In Figure I.3, a type of power- and ground-switched always-on buffer is shown. The cell has both power and ground rail connections, VSW and GSW, respectively, but they are not used by the cell. The actual power and ground pins the cell come from VDD and VSS, which need to be routed separately. The following command models this type of cells:

```
define_always_on_cell
   -cells LP_Buf_Pow_Gnd
   -power VDD -ground VSS
   -power switchable VSW -ground switchable GSW
```

The same command can also be used to describe any other type of power- and ground-switched always-on cells such as an inverter, AND gate, etc.



LP_Buf_Pow_Gnd

Figure I.3—Power- and ground-switched always-on buffer

I.1.5 Modeling a power-switched always-on flop with internal isolation

To model a power-switched always-on cell with internal isolation at some input pins, use the **define_always_on_cell** command (see $\underline{7.2}$) with the following options:

define_always_on_cell -cells cells -power pin -power_switchable pin -ground pin -isolated_pins list_of_pin_lists [-enable expression_list]

The always-on flip-flop cell in Figure I.4 has internal isolation at input pins SE and SI with the other input pin ISO as the control. The following command models this type of cells:

```
define_always_on_cell
    -cells LP_ff
```

```
-power VDD -power_switchable VSW -ground VSS \
-ioslated_pins { {SE SI} } -enable {!Iso}
```





Figure I.4—Power-switched always-on flop with input isolation on pins SE and SI

I.2 Modeling cells with internal diodes

Cells with input pins connected to diodes need to be properly modeled to avoid electrical failure in a design with power-management. To model such cells, use the **define_diode_clamp** command (see <u>7.3</u>) with the following options:

define_diode_clamp -cells cell_list -data_pins pin_list [-type <power | ground | both>] [-power pin] [-ground pin]

To describe the different type of diode connected pins shown in Figure I.5, use the following commands:

define_diode_clamp -cells cellA -data_pins in1 -type power -power VDD1
define_diode_clamp -cells cellB -data_pins in1 -type ground -ground VSS2
define_diode_clamp -cells cellC -data_pins in1 -type both \
 -power VDD1 -ground VSS2
define diode clamp -cells cellD -data pins in1 -type power -power VDD



Figure I.5—Cells with different type of internal diodes

I.3 Modeling isolation cells

This subclause shows examples for how to model various types of isolation cells.

I.3.1 Types of isolation cells

Isolation logic is required when the leaf-drivers and leaf-loads of a net are in power domains that are not on and off at the same time, or because it is part of the design intent. The following is a list of the most typical isolation cells:

- Isolation cell to be placed in the unswitched domain
- Isolation cell to be used in a ground-switchable domain
- Isolation cell to be used in a power-switchable domain
- Isolation cells to be used in a a power- or ground-switchable domain
- Isolation cells without follow pins that can be placed in any domain
- Isolation cells without always-on power pins that can be placed in a switchable power domain
- Isolation cells without an enable pin
- Isolation clamp cell
- Isolation level-shifter combo cell

All types of isolation cells are defined using the **define_isolation_cell** command (see <u>7.4</u>). The following subclauses indicate which command options to use for each type.

I.3.2 Modeling an isolation cell to be placed in the unswitched domain

To model an isolation cell to be placed in an unswitched domain, use the **define_isolation_cell** command (see <u>7.4</u>) with the following options:

```
define_isolation_cell

-cells cell_list

-power power_pin -ground ground_pin

-valid_location on

{-enable pin | -no_enable <high | low | hold>}
```

Figure I.6 shows an AND cell that can be used for isolation purposes.



Figure I.6—Dedicated isolation cell in unswitched domain

The following command models the isolation cell in Figure I.6:

```
define_isolation_cell \
   -cells IsoLL \
   -power VDD -ground VSS \
   -enable E \
   -valid_location on
```

NOTE—To use the cell in regular logic, add the **-non_dedicated** option. Non-dedicated cells are typically only placed in the unswitched domain (i.e., -valid_location on).

I.3.3 Modeling an isolation cell for ground-switchable domain

To model an isolation cell to be used in a ground-switchable domain, use the **define_isolation_cell** command (see <u>7.4</u>) with the following options:

define_isolation_cell
 -cells cell_list
 {-enable pin | -no_enable <high | low | hold>}
 -ground_switchable ground_pin
 -power power_pin -ground ground_pin
 [-valid_location <source | sink | on | off>]
 [-always_on_pins pin list]

Figure I.7 shows an AND cell that has the path from power to ground cut off on the ground side. This AND cell can only be used for isolation.

The following command models the isolation cell in <u>Figure I.7</u>, which can be placed at the output of a ground-switchable domain.

```
define_isolation_cell \
   -cells IsoLL \
   -ground_switchable GSW \
   -power VDD -ground VSS \
   -enable E \
   -valid location source
```



Figure I.7—Isolation cell with ground-switchable pin

I.3.4 Modeling an isolation cell for power-switchable domain

To model an isolation cell to be used in a power-switchable domain, use the **define_isolation_cell** command (see <u>7.4</u>) with the following options:

```
define_isolation_cell
    -cells cell_list
    {-enable pin | -no_enable < high | low | hold>}
    -power_switchable power_pin
    -power power_pin -ground ground_pin
    [-valid_location <source | sink | on | off>]
```

Figure I.8 shows an AND cell that has the path from power to ground cut off on the power side. This AND cell can only be used for isolation.





The following command models the isolation cell in Figure I.8.

```
define_isolation_cell \
  -cells IsoLL \
  -power_switchable VSW \
  -power VDD -ground VSS \
```

```
-enable E \
-valid location source
```

Such a cell would be a good candidate for an isolation strategy like the following, assuming PSW is a switchable domain.

set_isolation myIso -domain PSW -applies_to outputs \
 -isolation_signal iso -isolation_sense high \
 -clamp_value low -location self

1.3.5 Modeling an isolation cell for power- and ground-switchable domains

To model an isolation cell to be used in a power- and ground-switchable domain, use the **define_isolation_cell** command (see 7.4) with the following options:

```
define_isolation_cell
   -cells cell_list
   {-enable pin | -no_enable <high | low | hold>}
   -power_switchable power_pin -ground_switchable ground_pin
   -power power_pin -ground ground_pin
   [-valid_location <source | sink | on | off>]
   [-always_on_pins pin_list]
```

Figure I.9 shows an AND cell that has the path from power to ground cut off on the power and ground sides. This AND cell can only be used for isolation.



Figure I.9—Dedicated power- and ground-switchable isolation cell

The following command models the isolation cell in Figure I.9:

```
define_isolation_cell \
  -cells IsoLL \
  -power_switchable VSW -ground_switchable GSW \
  -power VDD -ground VSS \
  -enable E \
  -valid_location source
```

I.3.6 Modeling an isolation cell that can be placed in any domain

To model an isolation cell to be used in any domain, which typically does not have the power or ground rail connection, use the **define_isolation_cell** command (see 7.4) with the following options:

define_isolation_cell
 -cells cell_list
 {-enable pin | -no_enable <high | low | hold>}
 -power power_pin -ground ground_pin
 -valid_location any
 [-always_on_pins pin_list]

I.3.7 Modeling an isolation cell without always-on power pins that can be placed in a switchable power domain

In some cases, a regular single rail can also be placed at the output of a switchable domain and used for isolation. For example, for a 2-input NOR type cell, the output will be pull-down to the ground or logic zero as long as one of the inputs is logic one irrespective of the voltages at the power pins. As a result, such a cell can be placed within a power-gated domain to isolate the domain outputs to logic zero. To model such a cell, use the following command and options:

```
define_isolation_cell

-cells cell_list

-enable pin

-power_switchable power_pin -ground ground_pin

-valid_location off
```

Similarly, for a 2-input NAND type cell, the output will be driven to logic one as long as one of the inputs is logic zero, irrespective of the connection at the ground pins. As a result, such a cell can be placed within a ground-gated domain to isolate the domain outputs to logic one. To model such a cell, use the following command and options:

```
define_isolation_cell

-cells cell_list

-enable pin

-power power_pin -ground_switchable ground_pin

-valid_location off
```

Example

```
define_isolation_cell \
-cells NOR_ISO \
-power_switchable VDD -ground VSS \
-enable iso \
-valid location off
```

I.3.8 Modeling an isolation cell without enable pin

There are special isolation cells that do not have an enable pin, but still can clamp output to a logic value when the primary power supply is switched off. Such a cell looks like a buffer, but its functionality is different when the switchable power is on and off. These cells are useful to buffer a net that typically requires always-on buffers, e.g., the retention control pin of all retention flops. The advantage of using such a cell versus an always-on buffer is it consumes much less power. To model such a cell, use the **define_isolation_cell** command (see <u>7.4</u>) with the following options:

```
define_isolation_cell
-cells cell_list
-no_enable <high | low | hold>
```

```
[-power_switchable power_pin] [-ground_switchable ground_pin]
[-power power_pin] [-ground ground_pin]
[-valid_location <source | sink | on | off>]
[-always_on_pins pin_list]
```

Example

```
define_isolation_cell \
  -cells IsoLL \
  -power VDD -ground VSS \
  -no_enable low\
  -valid_location sink
```

I.3.9 Modeling an isolation clamp cell

An *isolation clamp high cell* is a simple PMOS transistor with the gate input being used as the enable pin. When its driver is switched off by a ground switch and the enable pin has value 0, the connected net can be clamped to a logic high value as shown in Figure I.10.



Figure I.10—Isolation clamp high cell

To model an isolation clamp high cell, use the **define_isolation_cell** command (see <u>7.4</u>) with the following options:

```
define_isolation_cell
-cells cell_list
-enable pin -clamp_cell high -power power_pin
-valid location on
```

An *isolation clamp low cell* is a simple NMOS transistor with the gate input being used as the enable pin. When its driver is switched off by a power switch and the enable pin has value 1, the connected net can be clamped to a logic low value as shown in <u>Figure I.11</u>



Figure I.11—Isolation clamp low cell

To model an isolation clamp low cell, use the **define_isolation_cell** command (see <u>7.4</u>) with the following options:

```
define_isolation_cell
-cells cell_list
-enable pin -clamp_cell low -ground ground_pin
-valid location on
```

Due to its special connectivity requirement, to apply such a power or ground clamp cell for a specific isolation strategy, use the **-port_map** option of the **use_interface_cell** command (see <u>6.55</u>). In terms of power and ground net connection, if it is a clamp low cell, only the isolation ground net specified in **-isolation_supply** is used; if it is a clamp high cell, only the isolation power net specified in **-isolation_supply** is used.

I.3.10 Modeling an isolation cell with multiple enable pins

Some isolation cells have an enable pin that is related to the non-switchable supply of the cell and additional enable pins that are related to the switchable supply. The switchable enable pin can be used to synchronize the isolation logic right before the non-switchable enable pin is activated or deactivated. To model an isolation cell with multiple enable pins, use the **define_isolation_cell** command (see <u>7.4</u>) with the following options:

```
define_isolation_cell
   -cells cell_list
   -aux_enables pin_list -enable pin [-clamp <high | low>]
   [-power_switchable power_pin] [-ground_switchable ground_pin]
   [-power power_pin] [-ground ground_pin]
   [-valid_location <source | sink | on | off | any>]
```

To specify an isolation strategy that targets these types of isolation cells, use the **set_isolation** command with the **-isolation_signal** option (see 6.41) by assigning a list of signals to the option. In this list, the first signal is the one to drive the enable pin and the rest of the signals drive the auxiliary enable pin specified in the **-aux_enables** option in the same order.

<u>Figure I.12</u> shows two examples of cells with multiple enable pins. The iso enable pin is related to the non-switchable supply vddc, while the en enable pin is related to the switchable supply vdd.



Figure I.12—Isolation cells with multiple enable pins

The following command models the isoandlow and isoorhigh cells in Figure I.12:

```
define_isolation_cell \
  -cells {isoandlow isoorhigh} \
  -aux_enables en \
  -power_switchable vdd \
  -power vddc -ground vss \
  -enable iso
```

The following commands show the isolation strategies that target the isoandlow and isoorhigh cells in Figure I.12:

```
set_isolation isol -domain PD1 -source PD1 \
    -isolation_signal { iso_drvr en_drvr} \
    -isolation_sense { high low } \
    -clamp_value 0
set_isolation iso2 -domain PD2 -source PD2 \
    -isolation_signal { iso_drvr en_drvr} \
    -isolation_sense { high high } \
    -clamp_value 1
```

I.3.11 Modeling a multi-bit isolation cell

A *multi-bit isolation cell* has multiple pairs of input and output pins with each pair serving as a single-bit isolation cell. An example is shown in Figure I.13.



Figure I.13—Multi-bit isolation cell

If the cell uses the same enable pin for all pairs of input and output pins, there is no difference in modeling such a multi-bit cell with respect to the single-bit isolation cell. If the cell has different enable pins for the input and output pairs, model the cell using the **define_isolation_cell** command with the **-pin_groups** option (see <u>7.4</u>).

The following command can be used to describe the multi-bit isolation cell for the power-switchable domain shown in <u>Figure I.13</u> (see <u>Figure I.8</u> for the corresponding single-bit cell):

```
define_isolation_cell -cells IsoLL \
    -power_switchable VSW \
    -power VDD -ground VSS \
    -pin_groups {{in1 out1 en1} {in2 out2 en2} {in3 out3 en3}}
```

I.4 Modeling level-shifters

This subclause shows examples for how to model various types of level-shifters.

I.4.1 Types of level-shifters

To pass signals between portions of the design that operate on different power or ground voltages, levelshifters are needed. The following is a list of the most typical level-shifters:

- Power level-shifters
- Ground level-shifters
- Enabled level-shifters
- Bypass level-shifters
- Multi-stage level-shifters
- Multi-bit level-shifters

All types of level-shifters are defined using the **define_level_shifter_cell** command (see <u>7.5</u>). The following subclauses indicate which command options to use for each type.

I.4.2 Modeling a power level-shifter

A power level-shifter passes signals between portions of the design that operate on different power voltages, but using the same ground voltages. To model a power level-shifter, use the following options from the **define_level_shifter_cell** command (see <u>7.5</u>):

define_level_shifter_cell
 -cells cell_list
 -input_voltage_range {{lower_bound upper_bound}*}
 -output_voltage_range {{lower_bound upper_bound}*}
 [-direction <low_to_high | high_to_low | both>]
 [-input_power_pin power_pin] [-output_power_pin]
 [-ground ground_pin]
 [-valid_location <source | sink | either | any>]

Figure I.14 shows a power domain at 0.8 V and one at 1.2 V. The ground voltage for both domains is 0.0 V. In this case, data signals going from the domain at 0.8 V to the domain at 1.2 V need a power level-shifter with direction low_to_high , while data signals going from the domain at 1.2 V to the domain at 0.8 V need a power level-shifter with direction high to low.



Figure I.14—Power level-shifter

The following commands can be used to model these power level-shifters:

```
define_level_shifter_cell -cells low_to_high_power \
    -input_voltage_range {{0.8 1.0}} -output_voltage_range {{1.0 1.2}} \
    -input_power_pin VDD_IN -output_power_pin VDD_OUT -ground VSS_IN \
    -direction low_to_high -valid_location source

define_level_shifter_cell -cells high_to_low_power \
    -input_voltage_range {{1.0 1.2}} -output_voltage_range {{0.8 1.0}} \
    -input_power_pin VDD_IN -output_power_pin VDD_OUT -ground VSS_IN \
    -direction high to low -valid location source
```

I.4.3 Modeling a ground level-shifter

A ground level-shifter passes signals between portions of the design that operate on different ground voltages, but using the same power voltages. To model a ground level-shifter, use the following options from the **define_level_shifter_cell** command (see 7.5):

```
define_level_shifter_cell
    -cells cell_list
    -ground_input_voltage_range {{lower_bound upper_bound}*}
    -ground_output_voltage_range {{lower_bound upper_bound}*}
    [-direction <low_to_high | high_to_low | both>]
```

[-input_ground_pin power_pin] [-output_ground_pin power_pin] [-power power_pin] [-valid_location <source | sink | either | any>]

The two power domains in Figure I.15 have the same power supply 1.2 V. However, the ground voltage for the first domain is at 0.0 V, while the ground voltage for the second domain is at 0.5 V. The direction of a level-shifter indicates the difference between the voltage swing of the driver and the voltage swing of the receiver. As a result, for data signals going from the domain with ground voltage 0.0 V to the domain with ground voltage 0.5 V, a ground level-shifter with direction high_to_low is required. Similarly, for data signals going from the domain with ground voltage 0.0 V, a ground level-shifter with direction high_to_low is required. Similarly, for data signals going from the domain with ground voltage 0.0 V, a ground level-shifter with direction low to high is required.



Figure I.15—Ground level-shifter

The following commands can be used to model these ground level-shifters:

```
define_level_shifter_cell -cells high_to_low_ground \
    -ground_input_voltage_range {{0.0 0.1}} \
    -ground_output_voltage_range {{0.4 0.5}} \
    -input_ground_pin VSS_IN -output_ground_pin VSS_OUT -power VDD_IN \
    -direction high_to_low -valid_location source
define_level_shifter_cell -cells low_to_high_ground \
    -ground_input_voltage_range {{0.4 0.5}} \
    -ground_output_voltage_range {{0.0 0.1}} \
    -input_ground_pin VSS_IN -output_ground_pin VSS_OUT -power VDD_IN \
    -direction low to high -valid location source
```

I.4.4 Modeling a power and ground level-shifter

A power and ground level-shifter passes signals between portions of the design that operate on different power and ground voltages. To model a ground level-shifter, use the following options from the **define_level_shifter_cell** command (see <u>7.5</u>):

```
define_level_shifter_cell
    -cells cell_list
    -input_voltage_range {{lower_bound upper_bound}*}
    -output_voltage_range {{lower_bound upper_bound}*}
    -ground_input_voltage_range {{lower_bound upper_bound}*}
    -ground_output_voltage_range {{lower_bound upper_bound}*}
    [-direction <low_to_high | high_to_low | both>]
    [-input_power_pin power_pin] [-output_ground_pin power_pin]
    [-input_ground_pin power_pin] [-output_ground_pin power_pin]
    [-valid_location <source | sink | either >]
```

The two power domains in Figure 1.16 have different power and ground voltages. domain_1 is the region where power is 0.8 V and ground is 0.5 V. domain_2 is the region where power is 1.2 V and ground is 0 V.

As shown, the voltage swing of the domain_1 is 0.3 V and the voltage swing of the domain_2 is 1.2 V. As a result, a low_to_high direction power and ground level-shifter is needed going from domain_1 to domain_2. Similarly, going from domain_2 to domain_1 requires a power and ground level-shifter in the high to low direction.



Figure I.16—Power and ground level-shifter

The following commands model the power and ground level-shifter to go from domain_1 to domain_2:

```
define_level_shifter_cell -cells low_to_high \
    -input_voltage_range {{0.8 1.0}} -output_voltage_range {{1.0 1.2}} \
    -ground_input_voltage_range {{0.4 0.5}} \
    -ground_output_voltage_range {{0.0 0.1}} \
    -input_ground_pin VSS_IN -output_ground_pin VSS_OUT \
    -input_power_pin VDD_IN -output_power_pin VDD_OUT \
    -direction low_to_high -valid_location source
```

The following commands model the power and ground level shift to go from domain 2 to domain 1:

```
define_level_shifter_cell -cells high_to_low \
    -input_voltage_range {{1.0 1.2}} -output_voltage_range {{0.8 1.0}} \
    -ground_input_voltage_range {{0.0 0.1}} \
    -ground_output_voltage_range {{0.4 0.5}} \
    -input_ground_pin VSS_IN -output_ground_pin VSS_OUT \
    -input_power_pin VDD_IN -output_power_pin VDD_OUT \
    -direction high_to_low -valid_location sink
```

I.4.5 Modeling an enabled level-shifter

An *enabled level-shifter* is the level-shifter with an enable pin, which allows the level-shifter to be used for isolation purpose in some cases. To model such a cell, use the **define_level_shifter_cell** command with the **-enable** option (see <u>7.5</u>).

This type of cell uses an enable pin to control the voltage shifting. Typically, the enable pin is related to the output supplies of the level-shifter. In other words, the enable control needs to have the same voltage as the receiving domain. If both domains are powered on, then the enable can be tied to a constant, such that the level-shifter is always active.

To model an isolation-level-shifter combo cell, see <u>I.4.9</u>.

I.4.5.1 Modeling an enabled power level-shifter

Assume the power level-shifter shown in <u>Figure I.14</u> also has an enable pin to enable the level-shifting functionality, as shown in <u>Figure I.17</u>.



Figure I.17—Enabled power level-shifter

In this cell, when the enable signal En is inactive (at logic 0), it protects the level-shifter cell when the input power supply is powered down and causes the output to be a specific logic value determined by its functionality. VLO and VSS are the primary power (low voltage) and ground pin, respectively, and VHI is the additional power pin (high voltage). As it is indicated by the primary power connection, the cell needs to be placed in the low-voltage domain. For such a cell to be used for isolation purposes when the driving domain is switched off using a header power switch, its input power pin needs to be connected to the primary power net of the driving domain because the driver of the level-shifter data pin is not protected, e.g., the inverter connected to A. In this case, the definition should be adjusted as follows:

```
define_level_shifter_cell -cells low_to_high_power_enable \
    -input_voltage_range {{0.8 1.0}} -output_voltage_range {{1.0 1.2}} \
    -input_power_pin VDD_IN -output_power_pin VDD_OUT -ground VSS_IN \
    -direction low_to_high -valid_location source \
    -enable En
```

The enable pin is related to the output supplies of the level-shifter.

I.4.5.2 Modeling an enabled ground level-shifter

Assume the ground level-shifter shown in <u>Figure I.15</u> also has an enable pin to enable the level-shifting functionality. VDD and VSS_IN are the primary power and ground pin (for higher ground voltage), respectively, and VSS_OUT is the additional ground pin (for normal ground voltage). The enable pin connection is analogous to the connection of the enabled power level-shifter in <u>Figure I.16</u>. In this case, the definition should be adjusted as follows:

```
define_level_shifter_cell -cells low_to_high_ground_enable \
    -ground_input_voltage_range {{0.4 0.5}} \
    -ground_output_voltage_range {{0.0 0.1}} \
    -input_ground_pin VSS_IN -output_ground_pin VSS_OUT -power VDD \
```

```
-direction low_to_high -valid_location source \
-enable en
```

The enable pin is related to the output supplies of the level-shifter.

I.4.6 Modeling a bypass level-shifter

To model a level-shifter whose level-shifting functionality can be bypassed under certain conditions, use the **define_level_shifter_cell** command with the **-bypass_enable** option (see 7.5).

An example of such a cell is shown in <u>Figure I.18</u>. When the bp_enable signal is *True*, the level-shifting functionality is bypassed and the signal OUT comes from the top buffer.





The following command can be used to describe a bypass level-shifter:

```
define_level_shifter_cell -cells low_to_high_mux \
    -input_voltage_range {{0.8 1.0}} -output_voltage_range {{1.0 1.2}} \
    -input_power_pin VDD_IN -output_power_pin VDD_OUT -ground VSS \
    -direction low_to_high -valid_location source -bypass_enable bp_enable
```

To apply such a cell for a specific level-shifter strategy, use the **-port_map** option of the **use_interface_cell** command (see <u>6.55</u>) to explicitly describe the pin connection for the bypass enable pin of the cell.

I.4.7 Modeling a multi-stage level-shifter

When the voltage difference between the driving (or originating) and receiving (or destination) power domains is large, multiple level-shifters or a single multi-stage level-shifter might be required. To model a single multi-stage level-shifter cell, define the level-shifter cell using the **define_level_shifter_cell** command with the **-multi_stage** option (see <u>7.5</u>) to identify the stage of the multi-stage level-shifter to which this definition (command) applies.

For a level-shifter cell with N stages, N definitions shall be specified for the same cell. Each definition needs to associate a number from 1 to N for this option to indicate the corresponding stage of this definition. A definition cannot have the same stage defined twice.

An example of a single multi-stage level-shifter cell is shown in Figure I.19.



Figure I.19—Multi-stage level-shifter

The following commands can be used to describe the single level-shifter cell shown in Figure 1.19:

```
define_level_shifter_cell -cells m_stage_ls -multi_stage 1 -input_power_pin
    V1\
-output_power_pin V2 -input_ground_pin VS1 -output_ground_pin VS2
define_level_shifter_cell -cells m_stage_ls -multi_stage 2 -input_power_pin
    V2\
-input_ground_pin VS2 -output_voltage_pin V3 -output_ground_pin VS2
```

To apply such a cell for a specific level-shifter strategy, use the **-port_map** option of the **use_interface_cell** command (see <u>6.55</u>) to explicitly describe the pin connections.

I.4.8 Modeling a multi-bit level-shifter cell

A multi-bit level-shifter cell has multiple pairs of input and output pins with each pair serving as a single-bit level-shifter. An example is shown in <u>Figure 1.20</u>.

For the following multi-bit level-shifter cells, there is no difference in modeling such a multi-bit cell with respect to a single-bit level-shifter cell:

- Multi-bit simple level-shifter without an enable pin
- Multi-bit enable level-shifter with the same enable pin for all bits

If the cell has different enable pins for the input and output pairs, model the cell using the **define_level_shifter_cell** command with the **-pin_groups** option (see <u>7.5</u>).

The following command can be used to describe the multi-bit level-shifter cell shown in Figure I.20:

```
define_level_shifter_cell -cells multi_bit_en \
    -input_voltage_range {{0.8 1.0}} -output_voltage_range {{1.0 1.2}} \
```

IEEE Std 1801-2013 IEEE STANDARD FOR DESIGN AND VERIFICATION OF LOW-POWER INTEGRATED CIRCUITS

```
-input_power_pin VDD_IN -output_power_pin VDD_OUT -ground VSS \
-direction low_to_high -valid_location source \
-pin groups {{in1 out1 en1} {in2 out2 en1} {in3 out3 en2}}
```



Figure I.20—Multi-bit level-shifter

I.4.9 Modeling an isolation level-shifter combo cell

A combo cell isolates or protects the input when the driving logic is powered down and generates an output isolation value at the same voltage as the output supply of the cell. Typically, the enable pin is related to the input supplies of the cell. The most common combo cells are the isolation cells with high-to-low shifting capabilities.

Modeling a combo cell requires two commands. For example, to model an isolation cell for powerswitchable domain that is also a power level-shifter, use the following definitions:

```
define_isolation_cell
   -cells cell_list
   {-enable pin | -no_enable <high | low | hold>}
   -power_switchable power_pin
   -power power_pin -ground ground_pin
   [-valid_location <source | sink>]

define_level_shifter_cell
   -cells cell_list
   -input_voltage_range {{lower_bound upper_bound}*}
   -output_voltage_range {{lower_bound upper_bound}*}
   -direction high_to_low
   [-input_power_pin power_pin] [-output_power_pin]
   [-yalid_location <source | sink>]
```

NOTE—The **-enable** option cannot be used in the **define_level_shifter_cell** definition. In addition, the same value for the **-valid_location** option needs to be specified in both the **define_isolation_cell** and **define_level_shifter_cell** commands.

To model an enabled level-shifter, see <u>1.4.5</u>.

I.5 Modeling power-switch cells

This subclause shows examples for how to model various types of power-switch cells.

I.5.1 Types of power-switch cells

To connect and disconnect the power (or ground) supply from the gates in internal switchable power domains, power-switch logic needs to be added. The following is a list of the most typical cells:

- Single-stage power-switch cell single transistor that controls the primary power supply to the logic of an internal switchable domain
- Single-stage ground-switch cell single transistor that controls the primary ground supply to the logic of an internal switchable domain
- Dual-stage power switch with a weak and strong transistor to control the primary power supply to the logic of an internal switchable domain
- Dual-stage ground switch with a weak and strong transistor to control the primary ground supply to the logic of an internal switchable domain

All types of power-switch cells are defined using the **define_power_switch_cell** command (see <u>7.6</u>). The following subclauses indicate which command options to use for each type.

I.5.2 Modeling a single-stage power-switch cell

To model a single-stage power-switch cell, use the following options from the **define_power_switch_cell** command (see <u>7.6</u>):

```
define_power_switch_cell
    -cells cell_list -type header
    -power_switchable power_pin -power power_pin
    -stage_1_enable expression [-stage_1_output expression]
    [-ground ground_pin]
    [-always_on_pins pin_list]
```

NOTE—The **-stage_1_output** and **-stage_1_ground** options do not need to be specified for an unbuffered powerswitch cell.

Figure I.21 shows a power-switch cell with an internal buffer. VIN is the pin connected to the unswitched power. VSW is the pin connected to the switchable power that is connected to the logic. When the enable signal Ei is activated, the unswitched power is supplied to the logic. As shown in Figure I.21, this type of cell usually contains a buffer that allows multiple power-switch cells to be chained together to form a power-switch column or ring. However, the power and ground of this buffer need to be unswitchable.



Figure I.21—Single-stage power switch

The following command models the power-switch cell shown in Figure I.21:

```
define_power_switch_cell -cells sw1 \
    -stage_1_enable Ei -stage_1_output Eo \
    -type header -power_switchable VSW -power VIN -ground VSS
```

I.5.3 Modeling a power-switch cell with gate bias

To model a single-stage power-switch cell with gate bias, use the following options from the **define_power_switch_cell** command (see $\underline{7.6}$):

```
define_power_switch_cell
    -cells cell_list -type header
    -gate_bias_pin power_pin
    -stage_1_enable expression [-stage_1_output expression]
    -power_switchable power_pin -power power_pin
    -ground ground pin [-always_on_pins pin list]
```

Typically, the enable pin is related to the power and the ground pin. With gate bias, the enable pin is typically related to the gate bias pin and the ground. The voltage on the gate bias pin is larger than the voltage of the power pin. Such a cell creates less leakage power compared to the cell without gate bias.

In <u>Figure 1.22</u>, the gate bias pin is VGB. Assume the input voltage VIN is at 1.2 V and the gate bias pin is at 3.3 V.



Figure I.22—Single-stage power switch with gate bias

The following command models the power-switch cell shown in Figure I.22:

```
define_power_switch_cell \
   -cells sw1 \
   -stage_1_enable Ei -stage_1_output Eo -gate_bias_pin VGB\
   -type header \
   -power_switchable VSW -power VIN -ground VSS
```

I.5.4 Modeling a single-stage ground-switch cell

To model a single-stage ground-switchable power-switch cell, use the following options from the **define_power_switch_cell** command (see $\underline{7.6}$):

```
define_power_switch_cell
    -cells cell_list -type footer
    -stage_1_enable expression [-stage_1_output expression]
    -ground_switchable ground_pin -ground ground_pin
    -power power_pin [-always_on_pins pin_list]
```

Figure I.23 shows a ground-switch cell. VSS is the pin connected to the unswitched ground. VSW is the pin connected to the switchable ground that is connected to the logic. When the enable signal Ei is activated, the unswitched ground is supplied to the logic. As shown in Figure I.23, this type of cell usually contains a buffer that allows multiple ground-switch cells to be chained together to form a ground-switch column or ring. However, the power and ground of this buffer need to be unswitchable.



Figure I.23—Single-stage ground switch

The following command models the ground-switch cell shown in Figure I.23:

```
define_power_switch_cell -cells gw1 \
    -stage_1_enable Ei -stage_1_output Eo \
    -type footer -ground_switchable GSW -ground VSS -power VDD
```

I.5.5 Modeling a dual-stage power-switch cell

To model a power-switch cell with two stages, use the following options from the **define_power_switch_cell** command (see $\underline{7.6}$):

```
define_power_switch_cell
    -cells cell_list -type header
    -power_switchable power_pin -power power_pin
```

-stage_1_enable expression [-stage_1_output expression] -stage_2_enable expression [-stage_2_output expression] -ground ground_pin [-always_on_pins pin_list]

Figure I.24 shows a dual-stage power-switch cell. VIN is the pin connected to the unswitched power. VSW is the pin connected to the switchable power that is connected to the logic. Only when both enable signals Ri and Ei are activated can the unswitched power be supplied to the logic. The Ri enable signal drives the stage-1 (weak) transistor, which requires less current to restore the unswitched power. The Ei enable signal drives the stage-2 (strong) transistor, which requires more current to fully supply the unswitched power to the logic. This type of cell usually contains two buffers that allow multiple power-switch cells to be chained together to form a power-switch column or ring. However, the power and ground of these buffers need to be unswitchable.



Figure I.24—Dual-stage power switch

The following command models the power-switch cell shown in Figure I.24:

```
define_power_switch_cell -cells sw1 \
    -stage_1_enable Ri -stage_1_output Ro \
    -stage_2_enable Ei -stage_2_output Eo \
    -type header -power switchable VSW -power VIN -ground VSS
```

I.5.6 Modeling a dual-stage ground-switch cell

To model a ground-switch cell with two stages, use the following options from the **define_power_switch_cell** command (see $\underline{7.6}$):

define_power_switch_cell -cells cell_list -type footer -ground_switchable ground_pin -ground ground_pin -stage_1_enable expression [-stage_1_output expression] -stage_2_enable expression [-stage_2_output expression] -power power_pin [-always_on_pins pin_list]

Figure I.25 shows a dual-stage ground-switch cell. VSS is the pin connected to the unswitched ground. GSW is the pin connected to the switchable ground that is connected to the logic. Only when both enable signals Ri and Ei are activated can the unswitched ground be supplied to the logic. The Ri enable signal drives the stage-1 (weak) transistor, which requires less current to restore the unswitched ground. The Ei enable signal drives the stage-2 (strong) transistor, which requires more current to fully supply the unswitched ground to the logic. This type of cell usually contains two buffers that allow multiple ground-switch cells to be chained together to form a ground-switch column or ring. However, the power and ground of these buffers need to be unswitchable.





The following command models the ground-switch cell shown in Figure 1.25:

```
define_power_switch_cell -cells gsw \
   -stage_1_enable Ri -stage_1_output Ro \
   -stage_2_enable Ei -stage_2_output Eo \
   -type footer -ground_switchable GSW -ground VSS -power VDD
```

I.6 Modeling state retention cells

This subclause shows examples for how to model various types of state retention cells.

I.6.1 Types of state retention cells

State retention cells are used for sequential cells to keep their previous state prior to power-down. The following is a list of the most typical state retention cells:

- State retention cell with explicit save control
- State retention cell with explicit restore control
- State retention cells with explicit save and restore controls
- State retention cells without explicit save or restore control

All types of state retention cells are defined using the **define_retention_cell** command (see <u>7.7</u>). The following subclauses indicate which command options to use for each type.

I.6.2 State retention cell that restores when power is turned on

To model a state retention cell that saves the current value when the control pin becomes active while the power is on, retains the saved value when power is off, and restores the saved value when the power is turned on, use the following options from the **define_retention_cell** command (see <u>7.7</u>):

```
define_retention_cell
   -cells cell_list [-cell_type string]
   -save_function {{pin < high | low | posedge | negedge}}
   [-always_on_pins pin_list]
   [-clock_pin pin]
   [-restore_check expression] [-save_check expression]
   [-retention_check expression] [-hold_check pin_list]
   [-always_on_components component_list]
   [-power_switchable power_pin] [-ground_switchable ground_pin]
   [-power power_pin] [-ground ground_pin]</pre>
```



Figure I.26 shows an example of such a cell.



To model the cell shown in Figure I.26, use the following command:

```
define_retention_cell -cells SR1 \
    -clock_pin Clk \
    -save_function {save posedge} \
    -restore_check !Clk -save_check !Clk \
    -power_switchable VDD_SW \
    -power VDD -ground VSS
```

If the UPF retention strategy is specified as follows:

```
set_retention ret -domain PD \
    -save_signal {save save_net posedge} \
    -restore_signal {save_net negedge} \
    ...
```

then the retention cells specified above are used to implement the strategy.

For a retention cell with output Q driven by a buffer powered by the retention supply (VDD), Q shall be specified in the **-always_on** option of the command, as follows:

```
define_retention_cell -cells SR1 \
    -clock_pin Clk \
    -always_on_pins {Q}
    -save_function {save posedge} \
    -restore_check !Clk -save_check !Clk \
    -power_switchable VDD_SW \
    -power VDD -ground VSS
```

Such a cell shall then be used to implement a retention strategy specified with **-use_retention_as_primary**, such as:

```
set_retention ret -domain PD \
    -save_signal {save save_net posedge} \
    -restore_signal {save_net negedge} \
    -use_retention_as_primary \
    ...
```

I.6.3 State retention cell that restores when control signal is deactivated

To model a state retention cell that saves the current value when the control pin becomes deactivated and restores the saved value when the control signal becomes activated, use the following options from the **define_retention_cell** command (see 7.7):

```
define_retention_cell
    -cells cell_list [-cell_type string]
    -restore_function {{pin < high | low | posedge | negedge}}
    [-always_on_pins pin_list]
    [-clock_pin pin]
    [-restore_check expression] [-save_check expression]
    [-retention_check expression] [-hold_check pin_list]
    [-always_on_components component_list]
    [-power_switchable power_pin] [-ground_switchable ground_pin]
    [-power power_pin] [-ground ground_pin]</pre>
```

Figure I.27 shows an example of such a cell.



Figure I.27—State retention with restore control

To model the cell shown in Figure I.27, use the following command:

```
define_retention_cell -cells SR1 \
   -clock_pin Clk \
   -restore_function {Ret negedge} \
   -power_switchable VDD_SW \
   -power VDD -ground VSS
```

If the UPF retention strategy is specified as follows:

```
set_retention ret -domain PD \
    -save_signal {save posedge} \
    -restore_signal {save negedge}
    ...
```

then the retention cells previously specified shall be used to implement the strategy.

Use **-restore_check**, **-save_check**, **-retention_check**, and **-hold_check** if the cell has additional requirements in retention mode.

In the previous example, if the clock signal needs to maintain low at the save and restore time, use the following command:

```
define_retention_cell -cells SR1 \
   -clock_pin Clk \
   -restore_function {Ret negedge} \
   -restore_check !Clk -save_check !Clk \
   -power_switchable VDD_SW \
   -power VDD -ground VSS
```

If the clock signal needs to also be low when the primary power is switched off, i.e., in retention mode, use the following command:

```
define_retention_cell -cells SR1 \
   -clock_pin Clk \
   -restore_function {Ret negedge} \
   -restore_check !Clk -save_check !Clk -retention check !Clk \
   -power_switchable VDD_SW \
   -power VDD -ground VSS
```

If the clock signal does not has to be low or high in at the save or restore, but it needs to maintain the same value before the cell entering retention mode and after the cell exiting retention mode, use the following command:

```
define_retention_cell -cells SR1 \
   -clock_pin Clk \
   -restore_function {Ret negedge} \
   -hold_check Clk \
   -power_switchable VDD_SW \
   -power VDD -ground VSS
```

I.6.4 State retention cells with save and restore controls

For a state retention cell with both save and restore controls, the cell saves the current value when the save control pin is activated and the power is on, while the cell restores the saved value when the restore control pin is activated. To model such a cell, use the following options from the **define_retention_cell** command (see <u>7.7</u>):

```
define_retention_cell
   -cells cell_list [-cell_type string] -save_function {{pin <high | low | posedge | negedge}}
   -restore_function {{pin <high | low | posedge | negedge}}
   [-always_on_pins pin_list] [-clock_pin pin]
   [-restore_check expression] [-save_check expression]
   [-retention_check expression] [-hold_check pin_list]
   [-always_on_components component_list]
   [-power_switchable power_pin] [-ground_ground_pin]
   [-power power_pin] [-ground ground_pin]</pre>
```

In this case, the cell saves the current value when the save expression is *True* and the power is on. The cell restores the saved value when the restore expression is *True* and the power is on. Figure I.28 shows an example of such a cell.





To model the cell shown in Figure I.28, use the following command:

```
define_retention_cell -cells SR2 \
    -clock_pin Clk \
    -restore_function {Wake high} -save_function {Sleep high} \
    -restore_check !Clk -save_check !Clk \
    -power_switchable VDD_SW \
    -power VDD -ground VSS
```

The state is saved when Sleep is active and the clock is down, and the state is restored when Wake is active and the clock is down.

If the UPF retention strategy is specified as follows:

```
set_retention ret -domain PD \
   -save_signal {save_net high} \
   -restore_signal {restore_net high}
   ...
```

then the retention cells previously specified shall be used to implement the strategy.

I.6.5 State retention cells without save or restore control

A master-slave type state retention cell does not have a dedicated save or restore control pin; it has a secondary power or ground pin to provide continuous power supply to the slave latch. Such a cell always saves a copy of the current value before entering the retention mode and the saved value is restored when the primary power is restore.

To model such a cell use the following **define_retention_cell** command options, without **-save_function** or **-restore_function**:

```
define_retention_cell
   -cells cell_list [-cell_type string]
   [-always_on_pins pin_list] [-clock_pin pin]
   [-restore_check expression] [-save_check expression]
   [-retention_check expression] [-hold_check pin_list]
   [-always_on_components component_list]
   [-power_switchable power_pin] [-ground_switchable ground_pin]
   [-power power_pin] [-ground ground_pin]
```

To specify a state retention strategy that targets these types of state retention cells, use the **set_retention** command (see 6.49) and do not use the **-save_signal** or **-restore_signal** options.

The following example models the master-slave retention cell ms ret:

```
define_retention_cell -cells ms_ret \
   -clock_pin CLK \
   -restore_check {!CLK} -save_check {!CLK}
```

The following command shows the state retention strategy that targets cell ms_ret for all registers with the power domain PD1:

```
set_retention sr1 -domain PD1 \
    -retention_condition {!clock && nreset} \
    -use_retention_as_primary \
    ....
```

Annex J

(normative)

Switching Activity Interchange Format

The Switching Activity Interchange Format (SAIF) is designed to assist in the extraction and storing of the switching activity information generated by electronic design automation (EDA) tools.

A SAIF file containing switching activity information can be generated using an HDL simulator and then the switching activity can be back-annotated into the power analysis/optimization tool as shown in <u>Figure J.1</u>. This type of SAIF file is called a *backward SAIF file*.



Figure J.1—Backward SAIF file

The power analysis/optimization tool, or some other EDA tool, may issue directives (instructions) to the backward SAIF file generation application on the format of the required SAIF file. These directives can be stored into a SAIF file, called a *forward SAIF file*, as shown in Figure J.2.

This annex provides the syntax and semantics of the backward SAIF file and the following two kinds of forward SAIF files:

- a) The library or gate-level forward SAIF file, which contains the directives for generating statedependent and path-dependent switching activity.
- b) The RTL forward SAIF file, which contains the directives for generating switching activity from the simulation of RTL hardware descriptions.



Figure J.2—Forward SAIF file

J.1 Syntactic conventions

The syntax of the SAIF file is described using the Backus-Naur Form (BNF), as follows:

Lowercase words (some containing underscores) are used to denote syntactic categories, e.g.,

backward_instance_info

Boldface words are used to denote the reserved keywords, operators, and punctuation marks that are a required part of the syntax, e.g.,

INSTANCE * ()

A non-boldface vertical bar (|) separates alternative items, e.g.,

binary_operator ::= * | ^ | |

Note that the last vertical bar is in **boldface** and therefore represents an actual operator rather than a separator between the alternative operators.

Non-boldface square brackets ([]) enclose optional items, e.g.,

date ::=
 (DATE [string])

Non-boldface braces ({}) enclose items that can be repeated 0 or more times, e.g.,

backward_saif_info ::=
 {backward_instance_info}

J.2 Lexical conventions

SAIF files are a stream of lexical tokens that consist of one or more characters. Except for one-line comments (see the following), the layout of SAIF files is free-format, i.e., spaces and newlines are only syntactically significant as token separators.

The following are types of lexical tokens in SAIF files:

- white space
- comments
- numbers
- strings
- parenthesis
- operators
- hierarchical separator character
- identifiers
- keywords

The rest of this subclause describes the lexical tokens used in SAIF files and their conventions.

J.2.1 White space

White spaces are sequences of spaces, tabs, newlines, and form-feeds. White spaces separate the other lexical tokens.

J.2.2 Comments

The SAIF format allows for both one-line comments and block comments. *One-line comments* start with the character sequence // and end with a newline. *Block comments* start with the character sequence /* and end with the first occurrence of the sequence */. Block comments are not nested.

J.2.3 Numbers

Numbers in SAIF files are either of the following:

- Non-negative decimal integers, which are represented by a sequence of decimal characters, e.g., 12, 012, or 1200.
- Non-negative real numbers, which are non-negative IEEE standard double-precision floating-point number representations, e.g., 1, 3.4, .7, 0.3, 2.4e2, or 5.3e-1.

J.2.4 Strings

A *string* in SAIF files is a possibly empty sequence of characters enclosed by double-quotes characters ("") and contained on a single line, e.g., "SAIF version 2.0" or "".

J.2.5 Parenthesis

Most of the constructs in SAIF files are enclosed between the left-parenthesis character (() and the right-parenthesis character ()).

J.2.6 Operators

An *operator* in SAIF files is one of the following characters: !, *, ^, and |. Operators are used in conditional expressions.

J.2.7 Hierarchical separator character

The *hierarchical separator* is a special character used in composing hierarchical port/pin/net/instance names from simple identifiers. The hierarchical separator character is defined in the header of SAIF files and can be either the / character or the . character.

J.2.8 Identifiers

A SAIF *identifier* is a non-empty sequence of alphanumeric characters, the underscore character () and escaped characters, followed by an optional decimal number enclosed in brackets ([]). *Escaped identifiers* consist of the \ character followed by a non-white space character. A SAIF identifier cannot start with a decimal digit (.) character and cannot contain the hierarchical separator character, unless it is escaped. The \ character used in an escaped character is not part of the identifier, so abc and $a \ b \ c$ represent the same identifiers. SAIF identifiers are case-sensitive, abc and ABC represent two different identifiers.

Examples

 $clk, clk_net, clk[4], clk\#4, clk\(4\), \lclk, or mod\/net$

where the hierarchical separator character is presumed to be /.

J.2.9 Keywords

A SAIF *keyword* is a special sequence of alphanumeric characters. SAIF keywords can be used as identifiers; to avoid possible ambiguity, escape the first character of identifiers that can be mistaken for keywords. SAIF keywords are case-sensitive. <u>Table J.1</u> shows the set of SAIF keywords.

COND	LEAKAGE	ТВ
COND_DEFAULT	LIBRARY	ТС
DATE	MODULE	TG
DESIGN	NET	TIMESCALE
DIRECTION	PORT	ТХ
DIVIDER	PROGRAM_NAME	TZ
DURATION	PROGRAM_VERSION	VENDOR
FALL	RISE	VIRTUAL_INSTANCE
IG	RISE_FALL	fs
ІК	SAIFILE	ms
INSTANCE	SAIFVERSION	ns
IOPATH	TO	ps
IOPATH_DEFAULT	T1	S
		us

Table J.1—SAIF keywords

J.2.10 Syntactic categories for token types

The syntax of the SAIF files described in this document use the syntactic categories shown in <u>Table J.2</u> for token types.

Syntactic category	Token type	
dnumber	Non-negative integer numbers	
rnumber	Non-negative real numbers	
string	Strings	
hchar	Possible hierarchical separator characters	
identifier	Simple (non-hierarchical) identifiers	
hierarchical_identifier	Hierarchical identifiers	

Table J.2—Token type categories

J.3 Backward SAIF file

This subclause describes the format of the *backward SAIF file*, which contains hierarchical instance-specific switching activity information.

J.3.1 SAIF file

The backward SAIF file consists of a left-parenthesis ((), the **SAIFILE** keyword, the backward SAIF header, the backward SAIF info, and a right-parenthesis ()), as shown in Syntax 1.

backward_saif_file ::=
 (SAIFILE backward_saif_header backward_saif_info)

Syntax 1—backward_saif_file

J.3.2 Header

Syntax 2 defines the backward SAIF file header.

backward saif header ::=	
backward_saif_version	
direction	
design_name	
date	
vendor	
program_name	
program_version	
hierarchy_divider	
time_scale	
duration	

Syntax 2—backward_saif_header

Each backward SAIF header construct is described in the following subclauses.

J.3.2.1 backward_saif_version

Syntax 3 defines the backward saif version.

backward_saif_version ::=
 (SAIFVERSION string)

Syntax 3—backward_saif_version

The string in this construct represents the version number of the SAIF file, i.e., 2.0.

J.3.2.2 direction

Syntax 4 defines the direction.

direction ::= (DIRECTION string)

Syntax 4—direction

The string in this construct represents the type of the SAIF file, i.e., backward.

J.3.2.3 design_name

Syntax 5 defines the design_name.

design_name ::= (DESIGN [string])

Syntax 5—design_name

The optional *string* in this construct represents the design for which the switching activity in the SAIF file has been generated.

J.3.2.4 date

Syntax 6 defines the date.

date ::= (DATE [string])

Syntax 6—date

The optional *string* in this construct represents the date the SAIF file was generated.

J.3.2.5 vendor

Syntax 7 defines the vendor.
vendor ::= (VENDOR [string])

Syntax 7—vendor

The optional *string* in this construct represents the name of the vendor whose application was used to generate the SAIF file.

J.3.2.6 program_name

Syntax 8 defines the program name.

program_name ::= (PROGRAM_NAME [string])

Syntax 8—program_name

The optional *string* in this construct represents the name of the application used to generate the SAIF file.

J.3.2.7 program_version

Syntax 9 defines the program_version.

```
program_version ::=
(PROGRAM_VERSION [string])
```

Syntax 9—program_version

The optional *string* in this construct represents the version number of the application used to generate the SAIF file.

J.3.2.8 hierarchy_divider

Syntax 10 defines the hierarchy divider.

hierarchy_divider ::=
 (DIVIDER [hchar])

Syntax 10—hierarchy_divider

The optional *hchar* in this construct represents the hierarchical separator character used in hierarchical identifiers. Only the / and . characters shall be specified as the hierarchical separator character; the default is the . character.

J.3.2.9 time_scale

Syntax 11 defines the time_scale.

time_scale ::=
 (TIMESCALE [dnumber timeunit])
timeunit ::=
 s | ms | us | ns | ps | fs

Syntax 11—time_scale

This construct specifies the units used for all time values in the SAIF file. The *dnumber* shall be 1, 10, or 100; it represents the scaling factor of the time values. For example, if the time scale of a SAIF file is

(TIMESCALE 100 us)

then all the time values in the SAIF file are specified in hundreds of microseconds. If the decimal number and time unit are not specified, the default time scale is 1 ns.

J.3.2.10 duration

Syntax 12 defines the duration.

duration ::= (DURATION rnumber)

```
Syntax 12—duration
```

This construct specifies the total time duration applied to the switching activity in the SAIF file.

J.3.2.11 Example

This is an example of a valid backward SAIF file header.

```
(SAIFVERSION "2.0")
(DIRECTION "backward")
(DESIGN "alu")
(DATE "Fri Jan 18 10:30:00 PDT 2002")
(VENDOR "SAIF'R'US Corp.")
(PROGRAM_NAME "saifgenerator")
(PROGRAM_VERSION "1.0")
(DIVIDER /)
(TIMESCALE 1 ns)
(DURATION 5000)
```

J.3.3 Simple timing attributes

This construct specifies the total duration (in time values) that some particular design net/port/pin (specified elsewhere) has some particular value. Syntax 13 defines this construct.

simple timing attribute ::=	
(T0 rnumber)	
(T1 rnumber)	
(TX rnumber)	
(TZ rnumber)	
(TB rnumber)	

Syntax 13—simple_timing_attribute

The different types of simple timing attributes are as follows:

- **T0** is the total time the design object has the value 0.
- **T1** is the total time the design object has the value 1.
- **TX** is the total time the design object has an unknown value.
- **TZ** is the total time the design object is in a floating bus state. A *floating bus state* is the state when all drivers on a particular bus are disabled and the bus has a floating logic value.
- **TB** is the total time the design object is in a bus contention state. A *bus contention state* is the state when two or more drivers simultaneously drive a bus to different logic levels.

Example

If the time scale is 100 μ s, then the following three simple timing attribute constructs:

(T0 100) (T1 92.5) (TX 7.5)

specify a particular design object has the value 0 for a total 10 000 μ s, the value 1 for a total of 9250 μ s, an unknown value for a total of 750 μ s, and it never reaches the floating bus and bus contention states.

J.3.4 Simple toggle attributes

This attribute construct specifies the number on a particular type of toggle registered on a particular design net/port/ pin (specified elsewhere). Syntax 14 defines this construct.



Syntax 14—simple_toggle_attribute

The different types of simple toggle attributes are as follows:

- TC is the number of 0 to 1 plus the number of 1 to 0 transitions. This is usually referred to as the *toggle count*.
- **TG** is the number of transport glitch edges (see J.3.4.1).
- IG is the number of inertial glitch edges (see J.3.4.2).
- IK is the inertial glitch de-rating factor. To estimate this factor, see <u>J.3.4.3</u>.

Example

The following simple toggle attributes:

(TC 200) (IG 6)

specify a total of 200 transitions between the 0 and 1 logic states, and a total of six inertial glitch edges are registered on some particular design object(s).

J.3.4.1 Transport glitch

Transport glitches are extra transitions at the output of the gate before the output signal reaches its steady state and, unlike inertial glitches (see <u>J.3.4.2</u>), can not be canceled by an inertial delay algorithm. A transport glitch consumes the same amount of power as a normal toggle transition and is an ideal candidate for power minimization during the optimization process. Transport glitches at the output of the gate have a pulse width longer than the gate delay and do not contribute to the functional behavior of the circuit.

In general, the number of transport glitch transitions occurring in the circuit is the difference between the total number of toggle transitions obtained from a full-timing simulation and that from a cycle-based simulation, assuming all inertial glitches (see <u>J.3.4.2</u>) have been filtered out by the timing simulator, i.e., the total number of toggles obtained from the timing simulator does not include inertial glitches. Figure J.3 shows a possible way to have transport glitches in the circuit. Although steady-state analysis of the circuit indicates that node N, the output of the XOR gate, should always remain at logic 1 regardless of the primary input, the additional timing delay due to the inverter causes a glitch at N whenever the input changes its state.



Figure J.3—Transport glitch

J.3.4.2 Inertial glitch

Inertial glitches are signal transitions occurring at the output of the gate, which can be filtered out if an inertial delay algorithm is applied. A simple example (see Figure J.4) best explains inertial glitches.



Figure J.4—Inertial glitch

A VHDL description for this inverter looks something like:

OUT ← not IN after 5 ns (inertial delay is implicitly presumed)

If the input pulse has a width less than 5 ns, the inertial delay algorithm shall cancel the signal transitions at the output of the inverter. However, some power is still consumed due to the two partial transitions at the output. Therefore, it is necessary to report these two inertial glitch transitions in a SAIF file.

NOTE—SAIF counts the number of glitches by signal edges, not signal pulses.

J.3.4.3 De-rating factor for inertial glitch

In <u>J.3.4</u>, glitching activities are categorized into two types, transport glitches and inertial glitches, and the number of glitch transitions are reported in the SAIF file. Transport glitches consume the same amount of power as normal toggles, so power consumption can be accurately calculated based on the number of transitions. For inertial glitches, however, the number of transitions is not enough to accurately estimate the inertial glitching power dissipation.

To improve the accuracy for inertial glitching power estimation, it is recommended that a simulator provide a de-rating factor for each node in the circuit that has inertial glitches. Described as follows, this de-rating factor can be used to scale the inertial glitch count to an effective count of normal toggle transition. Power analysis tools can use the adjusted inertial glitch count to improve estimation accuracy.

Assume a gate has a total number of k delays, with a delay value of T_i (i = 1...k) for each delay.

Define N_i (i = 1...k) as the total number of inertial glitch pulses due to the delay T_i , and δ_{ij} as the timing difference of the input events that cause glitch j ($j = 1...N_i$) due to the delay T_i .

Define N_e as the total number of inertial glitch edges of the gate. It is easy to see that N_i and N_e satisfy Equation (E.1).

$$\sum_{i=1}^{k} N_i = \frac{N_e}{2}$$
(E.1)

NOTE—The total number of the glitch pulses is half of the total number of the glitch edges.

With the parameters previously defined, a de-rating factor can be defined as shown in Equation (E.2).

$$K = 2 \times \frac{\sum_{i=j}^{k} \sum_{i=j}^{N_i} \frac{\delta_{ij}}{T_i}}{N_e}$$
(E.2)

Here is an example of how to use the de-rating factor. Consider again the example of the inverter shown in Figure J.5.



Figure J.5—Inverter

The power consumption at the output can be approximated as shown in Equation (E.3).

$$P = \frac{\delta}{T} \times 2 \times P0 \qquad 0 \le \delta \le T \tag{E.3}$$

where

- P0 is the power consumption of the gate during one normal full-level transition
- δ is the timing difference of the two input events that cause the glitch

T is the delay of the inverter

Equation (E.3) indicates that the inertial glitching power dissipation can be roughly modeled by the timing difference of the input events that causes the glitch and the delay of the gate beyond which there is no inertial glitch.

Accordingly, for a node with a total of N_i number of inertial glitch pulses due to the delay T_i (i = 1...k), the total power consumption can be estimated as shown in Equation (E.4).

$$P = \sum_{i=1}^{k} \sum_{j=1}^{N_i} \frac{\delta_{ij}}{T_i} \times 2 \times P0$$
(E.4)

Rearranging Equation (E.2) and substituting Equation (E.4), the power consumption can be simplified as shown in Equation (E.5).

$$P = K \times N_{\rm e} \times P0 \tag{E.5}$$

This suggests that the inertial glitching power can be calculated by converting the number of glitching transitions into the number of normal transitions by applying a de-rating factor.

J.3.5 State-dependent timing attributes

State-dependent timing attributes specify the time duration when a cell is in particular states. The *state* of a cell is defined as the logic value of its pins. Syntax 15 defines this construct.

```
state dep timing attributes ::=
    (state_dep_timing_item {state_dep_timing_item}
    [COND_DEFAULT sd_simple_timing_attributes])
state dep timing item ::=
    COND cond expr sd simple timing attributes
cond expr ::=
     port name
    | unary_operator cond expr
    cond expr binary operator cond expr
    (cond expr)
port name ::=
    identifier
unary_operator ::=
binary operator ::=
       ^ | |
sd_simple_timing_attributes ::=
    {sd simple timing attribute}
sd simple timing attribute ::=
     (T1 rnumber)
    (T0 rnumber)
```

Syntax 15—state_dep_timing_attributes

Here cond_expr represents conditional expressions on pin names; sd_simple_timing_attribute can only contain one of the following:

- T1 is the total time duration in which the cell is in any of its associated states.
- T0 is the total time duration in which the cell is not in any of its associated states.

A *conditional expression* specifies the set of states for which the condition holds. For example, given a cell with, three inputs, A, B, and C, and one output Y, the conditional expression

A | B

represents all the cell states when the input pin A is 1 or the input B is 1, while C and Y can have any value.

The precedence of the operators in conditional expressions is shown in the following sequence: ! (logical not), * (logical and), ^ (logical exclusive or), and | (logical or), where ! has the highest precedence.

A state-dependent timing attribute construct

```
(COND expr1 attrs1
COND expr2 attrs2
...
COND exprn attrsn
COND DEFAULT attrs default)
```

determines a priority-encoded specification of the timing attributes <code>attrs1,...,attrs_default</code>, i.e., the attributes <code>attrs1</code> apply for the set of states for which the condition <code>expr1</code> holds, while the attributes <code>attrs2</code> apply for the set of states where the condition <code>expr2</code> holds and <code>expr1</code> does not hold, etc. The attributes <code>attrs default</code> apply for all the states where none of the conditional expressions hold.

Example

The state-dependent timing attributes of the cell given in <u>Figure J.6</u> during the time duration given in the wave diagram in <u>Figure J.7</u> can be specified as follows:

```
(COND (A * B * Y) (T1 1) (T0 8)
COND (!A * B * Y) (T1 1) (T0 8)
COND (A * !(B * C)) (T1 2) (T0 7)
COND B (T1 1) (T0 8)
COND C (T1 1) (T0 8)
COND_DEFAULT (T1 3) (T0 6))
```



Figure J.6—A cell and its internal behavior



Figure J.7—A wave diagram

J.3.6 State-dependent toggle attributes

The toggle attributes on cell pins can be *state dependent*, i.e., the attributes are relevant only to particular cell states. Syntax 16 defines this construct.

<pre>state_dep_toggle_attributes ::= (state_dep_toggle_item {state_dep_toggle_item} [state_dep_default_toggle_item]) state_dep_toggle_item ::= COND cond_expr [(edge_type)] simple_toggle_attribute state_dep_default_toggle_item ::= COND_DEFAULT simple_toggle_attribute</pre>
COND_DEFAULT (edge_type) simple_toggle_attribute [COND_DEFAULT (edge_type) simple_toggle_attribute] edge_type ::= RISE FALL

Syntax 16—state_dep_toggle_attributes

Similar to state-dependent timing attributes, the state-dependent toggle attributes construct represents a priority-encoded attribute specification. The optional $edge_type$ is used to further differentiate the toggle count between 0 to 1 (**RISE**) and 1 to 0 (**FALL**) transitions.

The state-dependent toggle attributes construct can end with an optional **COND_DEFAULT** specification that has no edge restrictions. Otherwise, it can end with up to two **COND_DEFAULT** specifications having different edge restrictions.

Example

The following state-dependent toggle attributes construct

(COND A (RISE) (TC 20) COND A (FALL) (TC 15) COND B (RISE) (TC 5) COND B (FALL) (TC 10))

specifies a total toggle count of 50. Of the 25 rise transitions, 20 occur when pin A has a value of 1, and 5 occur when pin A has a value of 0 and B is 1. Of the 25 fall transitions, 15 occur when the pin A is 1, and 10 occur when the pin A is 0 and B is 1.

The state associated with an input pin transition is the cell state just before the time of the transition, e.g., in the wave diagram given in Figure J.8, the state associated with the rise transition on input pin A at time 10 is represented by the expression A * !B * !Y.

The state associated with an output pin transition is the cell state just before the time of the input pin transition, causing the output pin transition, e.g., in the wave diagram given in Figure J.8; the rise transition on the output pin Y at time 13 is caused by the rise transition on the input pin B at time 10. The state associated with the rise transition on Y is the cell state just before time 10 (not time 13). This state is represented by the expression !A * B * !Y.





J.3.7 Path-dependent toggle attributes

The toggle attributes on output cell pins can be *path dependent*, i.e., the attributes are relevant only to particular input pins causing the output toggles. Syntax 17 defines this construct.



Syntax 17—path_dep_toggle_attributes

Given a path-dependent toggle attributes construct

```
(IOPATH pins1 attrs1
IOPATH pins2 attrs2
...
IOPATH pinsn attrsn
IOPATH_DEFAULT attrs_default)
```

the attribute attrs1 represents toggles caused by the input pins in pins1, the attribute attrs2 represents toggles caused by the input pins in pins2, etc. The pin lists pins1, ..., pinsn are mutually exclusive. The attribute attrs_default represents toggles caused by the cell input pins not present in pins1, ..., pinsn. The pin lists pins1, ..., pinsn are also called the *path conditions* or *related pins*.

Example

The following path-dependent toggle attributes construct

```
(IOPATH A (TC 10)
IOPATH B (TC 20)
IOPATH C D (TC 5))
```

specifies a total of 35 toggle edges on a cell output port, of which 10 are caused by transitions on the input port A, 20 are caused by transitions on the input port B, and 5 are caused either by a transition on the input port C or D.

J.3.8 State- and path-dependent toggle attributes

The toggle attributes on output cell pins can be both state dependent and path dependent. The syntax of such toggle attributes is that of simple toggle attributes and path-dependent toggle attributes nested inside a state-dependent toggle attributes construct, as shown in Syntax 18.



Syntax 18—sdpd_toggle_attributes

Similarly to state-dependent toggle attributes and path-dependent toggle attributes, the SDPD toggle attributes construct represents a priority-encoded attribute specification.

Example

This is an example of an SDPD toggle attributes construct.

```
(COND A (RISE) (IOPATH B (TC 1))
COND A (FALL) (IOPATH B (TC 2))
COND B (RISE) (IOPATH A (TC 1))
COND B (FALL) (IOPATH A (TC 0))
COND_DEFAULT (RISE) (IOPATH A (TC 1)
IOPATH B (TC 0))
COND_DEFAULT (FALL) (IOPATH A (TC 0)
IOPATH B (TC 1)))
```

J.3.9 Net, port, and leakage switching specifications

The constructs for net, port, and leakage switching specification associate switching activity (given in terms of timing and toggle attributes) to individual design nets, ports, and cells.

J.3.9.1 Net switching specifications

The net switching specification construct associates switching activity to individual nets. Syntax 19 defines the backward net spec.

```
backward_net_spec ::=
    (NET backward_net_info {backward_net_info})
    backward_net_info ::=
        (net_name net_switching_attributes)
    net_name ::=
        identifier
    net_switching_attributes ::=
        {net_switching_attribute}
    net_switching_attribute ::=
        simple_timing_attribute
        | simple_toggle_attribute
```

Syntax 19—backward_net_spec

The switching attributes that can be associated to nets are simple timing attributes and simple toggle attributes.

Example

This is an example of a net switching specification assigning switching activity to the nets clk, rst, in1, in2, and out.

```
(NET
(clk (T0 100) (T1 100) (TC 50))
(rst (T0 180) (T1 20) (TC 2))
(in1 (T0 60) (T1 140) (TC 22))
(in2 (T0 80) (T1 120) (TC 12))
(out (T0 120) (T1 60) (TX 20) (TC 10))
)
```

J.3.9.2 Port switching specifications

The port switching specification construct associates switching activity to individual design ports and cell pins. Syntax 20 defines the backward port spec.

backward port spec ::=	
(PORT backward_port_inf	o {backward_port_info})
backward_port_info ::=	
(port_name port_switching	_attributes)
port_name ::=	
identifier	
port_switching_attributes ::=	
{port_switching_attribute}	
<pre>port_switching_attribute ::=</pre>	
simple_timing_attribute	
simple_toggle_attribute	
state_dep_toggle_attribute	s
path_dep_toggle_attribute	S
sdpd_toggle_attributes	

Syntax 20—backward_port_spec

The toggle attributes that can be associated to input cell pins can be simple or state dependent. The toggle attributes that can be associated to output cell pins can be simple, state dependent, path dependent, or both state and path dependent. The toggle attributes that can be associated to design ports have to be simple. The timing attributes that can be associated to design ports and cell pins have to be simple.

Example

This is an example of the port switching specification construct applied to an AND gate.

```
(PORT
(A (TO 8) (T1 7)
(COND B (RISE) (TC 1)
COND B (FALL) (TC 2)
COND DEFAULT (TC 1)))
(B (TO 9) (T1 6)
(COND A (RISE) (TC 2)
COND A (FALL) (TC 1)
COND DEFAULT (TC 3)))
(Y (TO 10) (T1 5)
(COND A (RISE) (IOPATH B) (TC 2)
COND A (FALL) (IOPATH B) (TC 1)
COND B (RISE) (IOPATH A) (TC 1)
COND B (FALL) (IOPATH A) (TC 2)
COND DEFAULT (TC 0)))
)
```

J.3.9.3 Leakage switching specifications

The leakage switching specification construct specifies the duration that a particular cell spends in particular states. This construct is a list of state-dependent timing attributes, as shown in Syntax 21.

```
backward_leakage_spec ::=
    (LEAKAGE state_dep_timing_attributes {state_dep_timing_attributes})
```

```
Syntax 21—backward_leakage_spec
```

Example

This is an example of a leakage switching specification.

```
(LEAKAGE
(COND (A * B) (T1 5) (T0 10))
COND (A | B) (T1 6) (T0 9))
(COND_DEFAULT (T1 4) (T0 11)))
```

J.3.10 Backward SAIF info and instance data

Design switching activity is organized hierarchically in the backward SAIF info construct (that follows the SAIF header in a backward SAIF file). The *backward SAIF info* is a list of backward instance info constructs, as shown in Syntax 22.

backward_net_spec backward_port_spec	<pre>backward_saif_info ::= {backward_instance_info} backward_instance_info ::= (INSTANCE [string] path {backward_instance_spec} {backward_instance_info}) (VIPTUAL_INSTANCE string path backward_pot_spec)</pre>
	backward_net_spec backward_port_spec

Syntax 22—backward_saif_info

backward_instance_info contains the switching activity of a particular cell or design instance. The optional *string* following the **INSTANCE** keyword is the cell/design name that is instantiated, and *path* is the hierarchical name of the actual instance. This is followed by a possibly empty list of instance switching specifications, which are the net, port, and leakage switching specifications described in <u>J.3.9</u>. For design instances, the instance info can recursively contain the switching activity of its sub-design and library cell instances.

backward_instance_info can also be used to specify the switching activity of cell instances where the port names of the instance are not known, e.g., in design flows where switching activity generated by RTL simulation is annotated to the synthesized gate-level netlist of the RTL design.

In this case, the *string* following the **VIRTUAL_INSTANCE** keyword represents the type of cell instance; it needs to be recognized by the application reading the backward SAIF file. The *path* represents the name of the instance, and backward_port_spec assigns switching activity to logical port names. The application reading the SAIF file needs to map the logical port names to the actual cell instance port names.

Example

For example, the following virtual instance construct

```
(VIRTUAL_INSTANCE "sequential" A_reg
(PORT
(Q (TO 220) (T1 370) (TC 122))
)
```

gives the switching activity of the positive output pin of a sequential element; the actual name of the output pin depends on the library cell that is used to implement the sequential cell, i.e., it can have a different name than Q.

J.4 Library forward SAIF file

The *Library forward SAIF file* contains the SDPD directives needed by simulators and other applications generating backward SAIF files that contain state-dependent and path-dependent switching activity. The SDPD directives can be generated from cell libraries with SDPD power characterization by using the appropriate tools.

For a description of state and path dependency, see <u>J.3</u>.

J.4.1 The SAIF file

The library forward SAIF file consists of a left-parenthesis ((), the **SAIFILE** keyword, the library forward SAIF header, the library forward SAIF info, and a finishing right-parenthesis ()), as shown in Syntax 23.

lforward_saif_file ::=
 (SAIFILE lforward_saif_header lforward_saif_info)

Syntax 23—Iforward_saif_file

J.4.1.1 Header

Syntax 24 defines the *library forward SAIF file header*.

lforward_saif_header ::= lforward_saif_version direction design_name date vendor program_name program_version hierarchy_divider

Syntax 24—Iforward_saif_header

Each library forward SAIF header construct is described in the following subclauses.

J.4.1.2 Iforward_saif_version

Syntax 25 defines the lforward_saif_version.

lforward_saif_version ::=
 (SAIFVERSION string [string])

Syntax 25—Iforward_saif_version

The first string in the this construct represents the version number of the SAIF file, i.e., 2.0.

The second *string* is optional and is either the string "lib" or "LIB"; this is used to specify that the SAIF file is a library forward SAIF file.

J.4.1.3 direction

Syntax 26 defines the direction.

direction ::= (DIRECTION string)

Syntax 26—direction

The *string* in the this construct represents the type of the SAIF file, i.e., forward.

J.4.1.4 design_name

Syntax 27 defines the design name.

design_name ::=
 (DESIGN [string])

Syntax 27—design_name

The optional *string* in this construct represents the design for which the forward SAIF file has been generated.

J.4.1.5 date

Syntax 28 defines the date.

```
date ::=
(DATE [string])
```

Syntax 28—date

The optional string in this construct represents the date the SAIF file was generated.

J.4.1.6 vendor

Syntax 29 defines the vendor.

vendor ::= (VENDOR [string])

Syntax 29-vendor

The optional *string* in this construct represents the name of the vendor whose application was used to generate the SAIF file.

J.4.1.7 program_name

Syntax 30 defines the program name.

```
program_name ::=
(PROGRAM_NAME [string])
```

Syntax 30—program_name

The optional string in this construct represents the name of the application used to generate the SAIF file.

J.4.1.8 program_version

Syntax 31 defines the program_version.

program_version ::=
 (PROGRAM_VERSION [string])

Syntax 31—program_version

The optional *string* in this construct represents the version number of the application used to generate the SAIF file.

J.4.1.9 hierarchy_divider

Syntax 32 defines the hierarchy divider.

hierarchy_divider ::=
 (DIVIDER [hchar])

Syntax 32—hierarchy_divider

The optional *hchar* in this construct represents the hierarchical separator character used in hierarchical identifiers. Only the / and . characters shall be specified as the hierarchical separator character; the default is the . character.

Example

This is an example of a valid library forward SAIF file header.

```
(SAIFVERSION "2.0" "lib")
(DIRECTION "forward")
(DESIGN)
(DATE "Fri Jan 18 10:00:00 PDT 2002")
(VENDOR "SAIFÍRÍUS Corp.")
(PROGRAM_NAME "libsaifgenerator")
(PROGRAM_VERSION "1.0")
(DIVIDER /)
```

J.4.2 State-dependent timing directive

State-dependent timing directives instruct the backward SAIF generator on the state conditions required in state-dependent timing attributes. Syntax 33 defines the state dep timing directive.

```
state_dep_timing_directive ::=
    (state_dep_timing_directive_item
    {state_dep_timing_directive_item}
    [COND_DEFAULT])
state_dep_timing_directive_item ::=
    COND cond_expr
```

Syntax 33—state_dep_timing_directive

A state-dependent timing directive is a list of directive items. The state-dependent timing attributes generated using such a timing directive contain switching activity assigned to a number of the states given in the directive. The order of any states in the timing attribute shall be the same as that in the timing directive.

Example

This is an example of a state-dependent timing directive.

```
(COND (A * B * C)
COND (!A * B * C)
COND (A * !(B * C))
COND B
COND C
COND DEFAULT)
```

J.4.3 State-dependent toggle directive

State-dependent toggle directives instruct the backward SAIF generator on the state and rise/fall conditions required in state-dependent toggle attributes. Syntax 34 defines the state_dep_toggle_directive.



Syntax 34—state_dep_toggle_directive

A state-dependent toggle directive is a list of directive items, each followed by an optional **RISE_FALL** keyword. The item list is followed by an optional **COND_DEFAULT** keyword, which can also be followed by an optional **RISE_FALL** keyword.

The state-dependent toggle attributes generated using such a toggle directive contain switching activity for a number of the states given in the directive. The order of any states in the toggle attribute shall be the same as that in the toggle directive. The **RISE_FALL** keyword instructs the backward SAIF generator that rise and fall edges can be differentiated and state-dependent toggle attribute items with **RISE** and/or **FALL** keywords can be generated.

Example

This is an example of a state-dependent toggle directive construct.

```
(COND (A*B) RISE_FALL
COND A RISE_FALL
COND B RISE_FALL
COND_DEFAULT)
```

J.4.4 Path-dependent toggle directive

Path-dependent toggle directives instruct the backward SAIF generator on the path conditions required in path-dependent toggle attributes for cell output pins. A *path condition* is a list of input port pins. Syntax 35 defines the path_dep_toggle_directive.

path_dep_toggle_directive ::=
 (path_dep_toggle_directive_item
 {path_dep_toggle_directive_item}
 [IOPATH_DEFAULT])
path_dep_toggle_directive_item ::=
 IOPATH port_name {port_name}

Syntax 35—path_dep_toggle_directive

A path-dependent toggle directive is a list of directive items. The path-dependent toggle attributes generated using such a toggle directive contain switching activity for a number of the path conditions (input pin lists) given in the directive. The order of the path conditions in the toggle attribute shall be the same as that in the toggle directive.

Example

This is an example of a path-dependent toggle directive construct.

(IOPATH A IOPATH B IOPATH C D)

J.4.5 SDPD toggle directives

SDPD toggle directives instruct the backward SAIF generator on the state and path conditions required in SDPD toggle attributes for cell output pins. The syntax of this construct is that of the path-dependent toggle directive embedded in the state-dependent toggle directive, as shown in Syntax 36.

```
sdpd_toggle_directive ::=
    (sdpd_toggle_directive_item {sdpd_toggle_directive_item}
    [COND_DEFAULT [RISE_FALL] [path_dep_toggle_directive]])
sdpd_toggle_directive_item ::=
    COND cond_expr [RISE_FALL] [path_dep_toggle_directive]
```

Syntax 36—sdpd_toggle_directive

The SDPD toggle attributes generated using such a toggle directive contain switching activity for a number of the state and path conditions given in the directive. The order of the conditions in the toggle attribute shall be the same as that in the toggle directive.

Example

This is an example of an SDPD toggle directive construct.

```
(COND A RISE_FALL (IOPATH B)
COND B RISE_FALL (IOPATH A)
COND_DEFAULT RISE_FALL
(IOPATH A
IOPATH B
IOPATH DEFAULT))
```

J.4.6 Module SDPD declarations

Module SDPD declarations instruct the backward SAIF generator on the type and structure of the required switching activity for particular cells. Syntax 37 defines this construct.

```
module_sdpd_declaration ::=
    (MODULE module_name {module_sdpd_directive})
module_name ::=
    identifier
module_sdpd_directive ::=
    port_declaration
    | leakage_declaration
    port_declaration ::=
        (PORT port_name {port_directive})
    port_directive ::=
        state_dep_toggle_directive
        | path_dep_toggle_directive
        | sdpd_toggle_directive
        leakage_declaration ::=
        (LEAKAGE {state_dep_timing_directive})
```

Syntax 37—module_sdpd_declaration

The module name *identifier* represents the library cell name.

A *port declaration* assigns port directives to the individual cell pins. Port directives are either statedependent toggle directives, path-dependent toggle directives, or SDPD toggle directives.

A *leakage declaration* consists of the **LEAKAGE** keyword followed by a state-dependent timing directive, which instructs the backward SAIF generator on the state conditions for the state-dependent timing attributes in backward leakage specifications.

Examples

This is an example of a port declaration.

```
(PORT
(A
(COND B RISE_FALL
COND_DEFAULT))
(B
(COND A RISE_FALL
COND_DEFAULT))
(Y
(COND A RISE_FALL (IOPATH B)
COND B RISE_FALL (IOPATH A)
COND_DEFAULT))
)
```

This is an example of a leakage declaration.

```
(LEAKAGE
(COND (A * B)
COND (A | B)
COND_DEFAULT)
)
```

J.4.7 Library SDPD information

The *SDPD declarations* for each library cell are listed in the library SDPD info constructs (that follow the SAIF header in the library forward SAIF file). Syntax 38 defines the library sdpd info.

library_sdpd_info ::=
 (LIBRARY string [string]
 {module sdpd declaration})

Syntax 38—library_sdpd_info

The first *string* following the **LIBRARY** keyword represents the name of the library. The second (optional) *string* sets the path of the directory containing the library and can be used for locating it.

J.5 RTL forward SAIF file

The *RTF forward SAIF file* lists the synthesis invariant points of an RTL design and provides a mapping from the RTL identifiers of these design objects to their synthesized gate-level identifiers. *Synthesis invariant points* are design objects (nets, ports, etc.) in the RTL description that are mapped directly to equivalent design objects in the synthesized gate-level descriptions. Examples of such points are the design ports and RTL identifiers (variables, signals, wires, etc.) that are mapped to the outputs of sequential cells.

J.5.1 SAIF file

The RTF forward SAIF file consists of a left-parenthesis ((), the **SAIFILE** keyword, the RTL forward SAIF header, the RTL forward SAIF info, and a finishing right-parenthesis ()), as shown in Syntax 39.

rforward_saif_file ::= (SAIFILE rforward_saif_header rforward_saif_info)

Syntax 39—rforward_saif_file

J.5.1.1 Header

Syntax 40 defines the *RTL forward SAIF file header*.

rforward_saif_header ::=
rforward_saif_version
direction
design_name
date
vendor
program_name
program_version
hierarchy_divider

Syntax 40—rforward_saif_header

Each RTL forward SAIF header construct is described in the following subclauses.

J.5.1.2 rforward_saif_version

```
Syntax 41 defines the rforward saif version.
```

rforward_saif_version ::= (SAIFVERSION string)

Syntax 41—rforward_saif_version

The *string* in the this construct represents the version number of the SAIF file, i.e., 2.0.

J.5.1.3 direction

Syntax 42 defines the direction.

direction ::= (DIRECTION string)

Syntax 42—direction

The *string* in the this construct represents the type of the SAIF file, i.e., forward.

J.5.1.4 design_name

Syntax 43 defines the design name.

design_name ::=
 (DESIGN [string])

Syntax 43—design_name

The optional *string* in this construct represents the design for which the forward SAIF file has been generated.

J.5.1.5 date

Syntax 44 defines the date.

date ::= (DATE [string])

Syntax 44—date

The optional *string* in this construct represents the date the SAIF file was generated.

J.5.1.6 vendor

Syntax 45 defines the vendor.

vendor ::= (VENDOR [string])

Syntax 45-vendor

The optional *string* in this construct represents the name of the vendor whose application was used to generate the SAIF file.

J.5.1.7 program_name

Syntax 46 defines the program_name.

program_name ::= (PROGRAM_NAME [string])

Syntax 46—program_name

The optional string in this construct represents the name of the application used to generate the SAIF file.

J.5.1.8 program_version

Syntax 47 defines the program version.

```
program_version ::=
(PROGRAM_VERSION [string])
```

Syntax 47—program_version

The optional *string* in this construct represents the version number of the application used to generate the SAIF file.

J.5.1.9 hierarchy_divider

```
Syntax 48 defines the hierarchy divider.
```

hierarchy_divider ::= (DIVIDER [hchar])

Syntax 48—hierarchy_divider

The optional *hchar* in this construct represents the hierarchical separator character used in hierarchical identifiers. Only the / and . characters shall be specified as the hierarchical separator character; the default is the . character.

Example

The following is an example of a valid library forward SAIF file header:

```
(SAIFVERSION "2.0")
(DIRECTION "forward")
(DESIGN "alu")
(DATE "Fri Jan 18 11:00:00 PDT 2002")
(VENDOR "SAIFÍRÍUS Corp.")
```

```
(PROGRAM_NAME "rtlsaifgenerator")
(PROGRAM_VERSION "1.0")
(DIVIDER /)
```

J.5.2 Port and net mapping directives

The *port and net mapping directives* in the RTL forward SAIF file contain a list of synthesis invariant port and net identifiers and their corresponding synthesized gate-level identifiers. Syntax 49 defines these constructs.



Syntax 49—Port and net mapping directives

Here, the rtl_name is mapped into the gate-level identifier mapped_name. Both the RTL name and mapped name in these constructs are represented by hierarchical identifiers.

In port_mapping_directives, the optional *string* is used for generating virtual instance data in the backward SAIF file and represents the type of the virtual instance.

J.5.3 Instance declarations

The port and net mapping directives in the RTL forward SAIF file are organized hierarchically in RTL forward instance declarations, which comprise the RTL forward SAIF instance info that follows the header in the forward SAIF file. Syntax 50 defines the RTL forward SAIF info constructs.



Syntax 50—RTL forward SAIF info constructs

The *RTL forward SAIF info* is a list of instance declarations. The optional *string* following the **INSTANCE** keyword represents the design name and the *hierarchical_identifier* following it is the actual instance name. The port and net mapping directives follow the instance name. The instance declarations of any sub-design instances can be included recursively in this construct.