



IEEE Standard for Design and Verification of Low Power Integrated Circuits

IEEE Computer Society

Sponsored by the
Design Automation Standards Committee

and the

IEEE Standards Association Corporate Advisory Group

IEEE
3 Park Avenue
New York, NY 10016-5997, USA
27 March 2009

IEEE Std 1801™-2009

IEEE Standard for Design and Verification of Low Power Integrated Circuits

Sponsor

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

Approved 19 March 2009

IEEE-SA Standards Board

Grateful acknowledgment is made to Accellera, Inc. for the permission to use the following source material:
Unified Power Format (UPF) Standard, Version 1.0

Abstract: The power supplied to elements in an electronic design affects the way circuits operate. Although this is obvious when stated, today's set of high-level design languages have not had a consistent way to concisely represent the regions of a design with different power provisions, nor the states of those regions or domains. This standard provides an HDL-independent way of annotating a design with power intent. In addition, the level-shifting and isolation between power domains may be described for a specific implementation, from high-level constraints to particular configurations. When the logic in a power domain receives different power supply levels, the logic state of portions of the design may be preserved with various state-retention strategies. This standard provides mechanisms for the refined and specific description of intent, effect, and implementation of various retention strategies. Incorporating components into designs is greatly assisted by the encapsulation and specification of the characteristics of the power environment of the design and the power requirements and capabilities of the components; this information encapsulation mechanism is also described in this standard. The analysis of the various power modes of a design is enabled with a combination of the description of the power modes and the collection, generation, and propagation of switching information.

Keywords: corruption semantics, interface specification, IP reuse, isolation, level-shifting, power-aware design, power intent, power domains, power modes, power states, progressive design refinement, retention, retention strategies

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

Copyright © 2009 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 27 March 2009. Printed in the United States of America.

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

PDF: ISBN 978-0-7381-5929-4 STD95919
Print: ISBN 978-0-7381-5930-0 STDPD95919

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.

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. The 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 the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the 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. The 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 this, or any other IEEE Standard document.

The IEEE does not warrant or represent the accuracy or content of the material contained herein, 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 herein 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 five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, 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 this document available, the IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is the IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing this, and any other IEEE Standards document, should rely upon his or her 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.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of concerned interests, it is important to ensure that any interpretation has also received 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 interpretation requests except in those cases where the matter has previously received formal consideration. 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 interpretation of the 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, explanation, or interpretation of the IEEE.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Comments on standards and requests for interpretations should be submitted to the following address:

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

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.

Introduction

This introduction is not part of IEEE Std 1801-2009, 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/IEEE 61691-1-1)^a and Verilog (IEEE Std 1364[™]) [B2]^b 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 90–100 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](#).

^bThe number in brackets correspond to those of the bibliography in [Annex A](#).

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, 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. The donation was executed to the P1801 working group and, although this standard is the first version of what is formally titled the IEEE Standard for the Design and Verification of Low Power Integrated Circuits, it represents the second version of what is more colloquially referred to as UPF.

Notice to users

Laws and regulations

Users of these documents should consult all applicable laws and regulations. Compliance with the provisions of this standard does not imply compliance to any applicable regulatory requirements. Implementers of the standard are responsible for observing or referring to the applicable 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 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 Standards Association Web site at <http://ieeexplore.ieee.org/xpl/standards.jsp>, or contact the IEEE at the address listed previously.

For more information about the IEEE Standards Association or the IEEE standards development process, visit the IEEE-SA website at <http://standards.ieee.org>.

Errata

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

Interpretations

Current interpretations can be accessed at the following URL: <http://standards.ieee.org/reading/ieee/interp/index.html>.

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 with respect to the existence or validity of any patent rights in connection therewith. A patent holder or patent applicant has filed a statement of assurance that it will grant licenses under these 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. Other Essential Patent Claims may exist for which a statement 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:

Stephen Bailey, *Chair*
Gary Delp, *Vice-Chair*
Joe Daniels, *Technical Editor*

Karen Bartleson
Frank Berntsen
Jason Binney
John Biggs
Minh Chau
Marc Edwards
Ed Huijbregts

Knut Just
Juergen Karmann
Kevin Kranen
Rolf Lagerquist
Olivier Lunven
Lisa McIlwain
Arvind Narayanan

Judith Richardson
Michael Rifani
Arturo Salz
Andrew Saunders
Eike Schmidt
Jim Sproch
Yatin Trivedi

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

ARM
Accellera
IBM
Improv Systems

Intel
JEITA
LSI Corporation
Magma Design Automation

Mentor Graphics
NXP
Synopsys
Texas Instruments

When the IEEE-SA Standards Board approved this standard on 19 March 2009, it had the following membership:

Robert M. Grow, *Chair*

Steve M. Mills, *Past Chair*

Judith Gorman, *Secretary*

John Barr
Karen Bartleson
Victor Berman
Ted Burse
Richard DeBlasio
Andy Drozd
Mark Epstein

Alexander Gelman
Jim Hughes
Rich Hulett
Young Kyun Kim
Joseph L. Koepfinger*
John Kulick
David Law
Ted Olsen

Glenn Parsons
Ron Petersen
Chuck Powers
Thomas Prevost
Narayanan Ramachandran
Jon Rosdahl
Sam Sciacca

*Member Emeritus

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

Satish K. Aggarwal, *NRC Representative*
Michael Janezic, *NIST Representative*
Howard Wolfman, *TAB Representative*

Michelle Turner
IEEE Standards Program Manager, Document Development

Michael D. Kipness
IEEE Standards Program Manager, Technical Program Development

Noelle Humenick
IEEE Standards Corporate Client Manager

Contents

1.	Overview	1
1.1	Scope	1
1.2	Purpose	1
1.3	Key characteristics of the Unified Power Format (UPF)	1
1.4	Power supply network design intent	3
1.5	Extending logic specification	5
1.6	Conventions used	6
1.7	Use of color in this standard	7
1.8	Contents of this standard	7
2.	Normative references	9
3.	Definitions, acronyms, and abbreviations	9
3.1	Definitions	9
3.2	Acronyms and abbreviations	12
4.	Power domains, supply sets, name spaces, and precedence	15
4.1	Power domains	15
4.2	Supply nets and ports	16
4.3	Supply sets	16
4.3.1	Explicit connection of supply nets	17
4.3.2	Automatic connection of supply nets	17
4.3.3	Implicit connection of supply nets	17
4.3.4	Predefined supply set functions	18
4.4	Naming rules	18
4.5	Name space semantics	19
4.6	Attributes and HDLs	20
4.7	Precedence	21
4.8	Lexical elements	21
4.9	Units	21
4.10	Boolean expressions	22
5.	Simulation semantics	23
5.1	Supply network creation	23
5.2	Supply network simulation semantics	23
5.2.1	Supply network initialization	23
5.2.2	Supply network update and evaluation	24
5.3	Power switch modeling	24
5.4	Power states	26
5.4.1	Power states of supply nets and ports	26
5.4.2	Power states of supply sets	26
5.4.3	Power states of power domains	27
5.4.4	Power states of systems and subsystems	31
5.5	Power state name spaces	32
5.6	Simstate simulation semantics	32

5.6.1	NORMAL	33
5.6.2	CORRUPT	33
5.6.3	CORRUPT_ON_ACTIVITY	33
5.6.4	CORRUPT_STATE_ON_CHANGE	33
5.6.5	CORRUPT_STATE_ON_ACTIVITY	34
5.6.6	NOT_NORMAL	34
5.7	Transitioning from one simstate state to another	34
5.7.1	Any state transition to CORRUPT	34
5.7.2	Any state transition to CORRUPT_ON_ACTIVITY	34
5.7.3	Any state transition to CORRUPT_STATE_ON_CHANGE	34
5.7.4	Any state transition to CORRUPT_STATE_ON_ACTIVITY	35
5.7.5	Any state transition to NORMAL	35
5.7.6	Any state transition to NOT_NORMAL	35
6.	Commands	37
6.1	Conventions used	37
6.2	Generic UPF command semantics	38
6.3	effective_element_list semantics.....	38
6.3.1	Transitive TRUE	39
6.3.2	Result	40
6.4	Command refinement.....	41
6.5	Error handling	42
6.5.1	errorCode	42
6.5.2	errorInfo	43
6.6	add_domain_elements.....	43
6.7	add_port_state	44
6.8	add_power_state	45
6.9	add_pst_state	47
6.10	associate_supply_set	48
6.11	bind_checker	50
6.12	connect_logic_net	51
6.13	connect_supply_net	52
6.14	connect_supply_set	53
6.15	create_composite_domain	55
6.16	create_hdl2upf_vct	56
6.17	create_logic_net	58
6.18	create_logic_port	58
6.19	create_power_domain	59
6.20	create_power_switch	61
6.21	create_pst	64
6.22	create_supply_net	65
6.22.1	Supply net resolution	65
6.22.2	Resolutions methods	66
6.22.3	Supply nets defined in HDL	67
6.23	create_supply_port	67
6.24	create_supply_set	68
6.24.1	Predefined supply set functions	69
6.24.2	Referencing supply set functions	69
6.25	create_upf2hdl_vct	70
6.26	describe_state_transition	71
6.27	load_simstate_behavior	72
6.28	load_upf	72
6.29	load_upf_protected	73

6.30	map_isolation_cell	74
6.31	map_level_shifter_cell	76
6.32	map_power_switch	77
6.33	map_retention_cell	78
6.34	merge_power_domains	81
6.35	name_format	83
6.36	save_upf	84
6.37	set_design_attributes	85
6.38	set_design_top	86
6.39	set_domain_supply_net	86
6.40	set_isolation	88
6.41	set_isolation_control	94
6.42	set_level_shifter	95
6.43	set_partial_on_translation	100
6.44	set_pin_related_supply	101
6.45	set_port_attributes	102
6.46	set_power_switch	106
6.47	set_retention	108
6.48	set_retention_control	112
6.49	set_retention_elements	114
6.50	set_scope	115
6.51	set_simstate_behavior	115
6.52	upf_version	116
6.53	use_interface_cell	117
7.	Queries	121
7.1	find_objects	122
7.1.1	Pattern matching and wildcarding	123
7.1.2	Wildcarding examples	123
7.2	query_upf	124
7.3	query_associate_supply_set	126
7.4	query_bind_checker	127
7.5	query_cell_instances	128
7.6	query_cell_mapped	128
7.7	query_composite_domain	129
7.8	query_design_attributes	130
7.9	query_hdl2upf_vct	131
7.10	query_isolation	132
7.11	query_isolation_control	133
7.12	query_level_shifter	135
7.13	query_map_isolation_cell	136
7.14	query_map_level_shifter_cell	137
7.15	query_map_power_switch	138
7.16	query_map_retention_cell	139
7.17	query_name_format	140
7.18	query_net_ports	141
7.19	query_partial_on_translation	142
7.20	query_pin_related_supply	142
7.21	query_port_attributes	143
7.22	query_port_direction	144
7.23	query_port_net	144
7.24	query_port_state	145
7.25	query_power_domain	146

7.26	query_power_domain_element	147
7.27	query_power_state	147
7.28	query_power_switch	148
7.29	query_pst	149
7.30	query_pst_state	150
7.31	query_retention	151
7.32	query_retention_control	153
7.33	query_retention_elements	154
7.34	query_simstate_behavior	155
7.35	query_state_transition	156
7.36	query_supply_net	157
7.37	query_supply_port	158
7.38	query_supply_set	159
7.39	query_upf2hdl_vct	160
7.40	query_use_interface_cell	161
8.	Switching Activity Interchange Format (SAIF)	163
8.1	Syntactic conventions.....	164
8.2	Lexical conventions.....	165
8.2.1	White space	165
8.2.2	Comments	165
8.2.3	Numbers	165
8.2.4	Strings	166
8.2.5	Parenthesis	166
8.2.6	Operators	166
8.2.7	Hierarchical separator character	166
8.2.8	Identifiers	166
8.2.9	Keywords	166
8.2.10	Syntactic categories for token types	167
8.3	Backward SAIF file.....	167
8.3.1	SAIF file	168
8.3.2	Header	168
8.3.3	Simple timing attributes	171
8.3.4	Simple toggle attributes	171
8.3.5	State-dependent timing attributes	173
8.3.6	State-dependent toggle attributes	175
8.3.7	Path-dependent toggle attributes	177
8.3.8	SDPD toggle attributes	178
8.3.9	Net, port, and leakage switching specifications	178
8.3.10	Backward SAIF info and instance data	180
8.4	Library forward SAIF file	181
8.4.1	The SAIF file	182
8.4.2	State-dependent timing directive	184
8.4.3	State-dependent toggle directive	185
8.4.4	Path-dependent toggle directive	185
8.4.5	SDPD toggle directives	186
8.4.6	Module SDPD declarations	186
8.4.7	Library SDPD information	187
8.5	The RTL forward SAIF file	188
8.5.1	The SAIF file	188
8.5.2	Port and net mapping directives	190
8.5.3	Instance declarations	191

Annex A (informative) Bibliography	193
Annex B (normative) Supply net logic type	195
Annex C (normative) Value conversion tables (VCTs)	207
Annex D (informative) UPF procs	211
Annex E (informative) De-rating factor for inertial glitch	217

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. 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).

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 (UPF)

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 a hardware description language (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 [Table 1](#) and [Annex B](#)). UPF also defines consistent semantics across verification and implementation, i.e., what is implemented is the same as what has been verified.

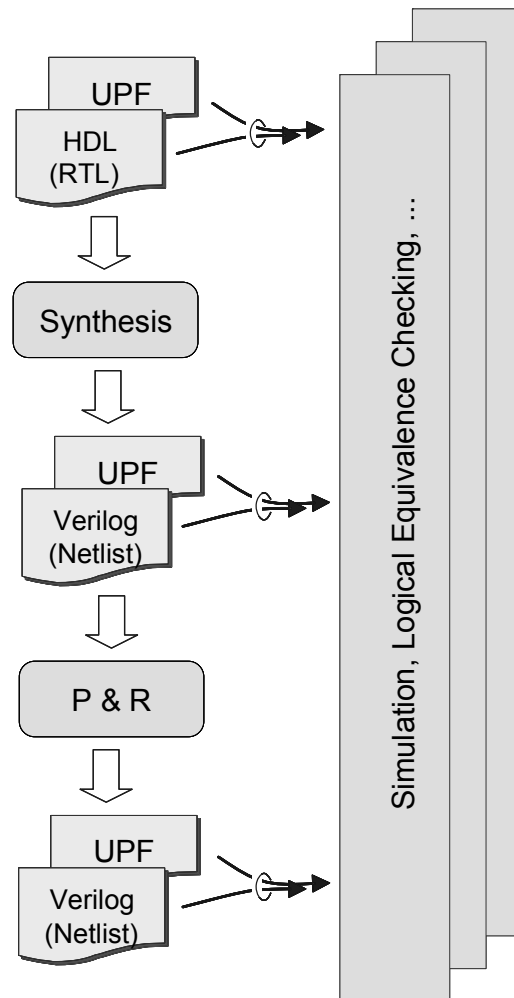


Figure 1—UPF tool flow

As indicated in [Figure 1](#), UPF files are part of the design source. 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.

- 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 names change, a UPF file with the new names is needed. A UPF-aware logical equivalence checker can read the full design filesets and perform the checks including the results of the UPF commands to ensure equivalence.

- Place and Route tools read both the netlist and the UPF files and produce outputs, potentially including an output UPF file.

UPF is a concise power design intent specification capability. Power design intent can be easily specified over many elements in the design. A UPF specification can be included with the other deliverables of intellectual property (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.

A UPF *specification* defines how to create a supply network to supply power to each design element, 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 design elements. By controlling the operating voltages of each supply net and whether the supply nets (and their connected design elements) are turned on or off, the supply network only provides power at the level the functional areas of the chip need to complete the computational task in a timely manner.

1.4 Power supply network design intent

Designing electronics to meet low power design constraints requires the specification of a power supply network that can control the distribution of that supply to minimize energy consumption. UPF supports the specification of the power supply distribution network so the supply network can be automatically implemented at a relatively abstract level.

To help manage the complexity of the supply network specification, *power domains* are defined to group elements from the logic hierarchy that share common supply needs. By default, all logic elements in a power domain use the same primary supply. Additional supplies may be defined to serve different uses in a power domain. In addition to the primary supply, UPF provides well-defined semantics for other supplies for both verification and implementation contexts.

The *supply network* consists of supply ports, switches, and supply nets. Supply network objects are defined within the logical hierarchy relative to the context of a power domain. *Supply ports* provide the supply interface to the logical hierarchy and the power domain's elements. Supply ports also provide the supply interface to switches. *Switches* control the supply distribution. *Supply nets* connect supply ports.

Although there is an obvious inference to the actual wires and ports in the implemented hardware, a UPF supply network is an abstraction of the electrical network on the chip. UPF defines no routing or layout information. As the supply network is specified apart from the logic design, the logic design specification remains independent of a specific power supply network specification.

[Figure 2](#) shows an example supply distribution network for a hypothetical chip, mySoC. A top-level power domain, mySoC_PD defined at the U_top instance level in the logical hierarchy, is defined with three supply ports, Pbat, Pwall, and GND. These ports represent the off-chip power sources. A single switch (S1) controls whether the chip receives its supply from Pbat or Pwall. Supply nets connect the output from S1 to each power domain defined within the top-level power domain. mySoC_PD does not contain any logic elements other than the root of the design U_top; its purpose is to define the interface to the off-chip power sources and provide the top-level supply network.

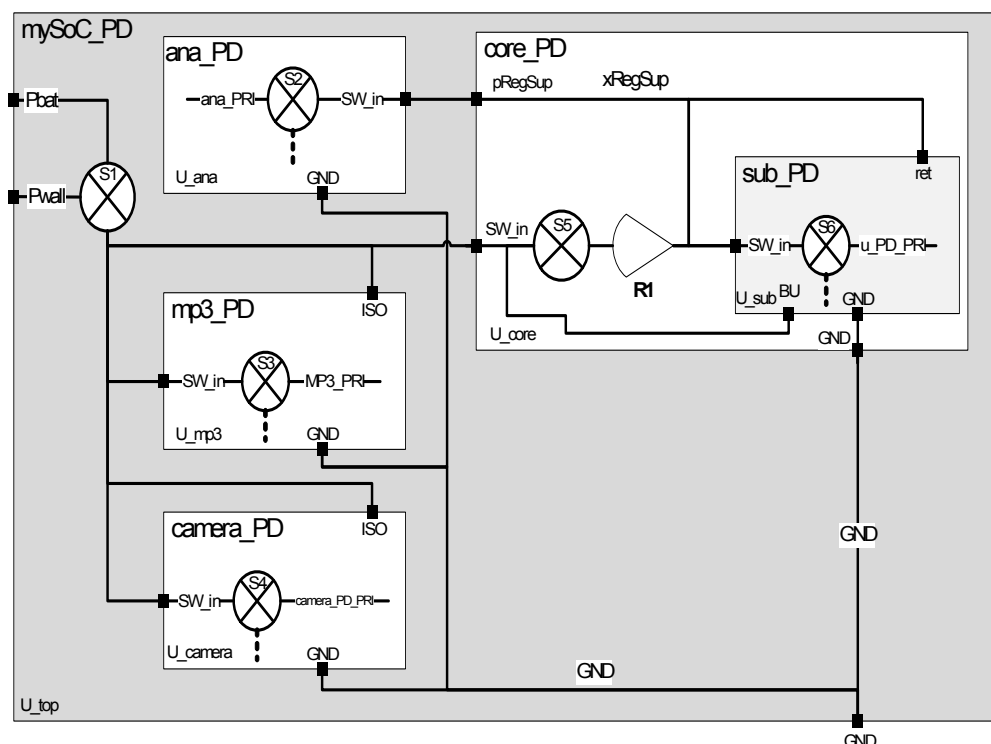


Figure 2—A supply distribution network

Four power domains are defined at the next level, `ana_PD`, `camera_PD`, `core_PD`, and `mp3_pd`. `camera_PD`, `mp3_PD`, and `ana_PD` each have a similar supply network internal to them. The primary supply is switched (S_4 , S_3 , and S_2), which indicates power to the logic elements in the domains can be turned off. Unswitched isolation supplies are provided for `camera_PD` and `mp3_PD`.

`core_PD` contains a nested domain `sub_PD`. `core_PD`'s primary supply `xRegSup` is switched and regulated within the domain, as it is the output of a regulator that has a switch (S_5) driving the regulator's input (R_1). Besides being the primary supply for `core_PD`, `xRegSup` also provides a retention supply for `sub_PD`. The regulated supply is also connected to the output supply port (`pRegSup`) and, ultimately, the switch input (S_2) whose output drives the primary supply for `ana_PD`. This indicates `ana_PD` is powered on only when `core_PD` is powered on. Within `sub_PD`, the primary supply is switched (S_6), indicating `sub_PD` can be powered down separately from `core_PD`, but holding the relationship that if `core_PD` is powered down, so is `sub_PD`. `sub_PD` also contains two additional supplies (`ret` and `BU`). The `ret` supply (which is used as a retention supply) has the same source as `core_PD`'s primary supply (the output from switch S_5) and the `BU` supply has the same source as the input to switch S_5 ; therefore, the retention supply (`ret`) is valid only when `core_PD` is on (switch S_5 in the `core_PD` power domain). The `BU` supply is provided for any retention elements whose state needs to be saved when both `sub_PD` and `core_PD` are powered down.

This example demonstrates a methodology of locating the switches for a power domain's supplies within the power domain. UPF also supports placing the switches that can be switched off outside the power domain. If this methodology were being used, switches S_2 , S_3 , and S_4 would be located in `mySoC_PD` and additional supply nets to connect the outputs of the switches to the corresponding ports of the nested power domains' logical hierarchies would be created. Similarly, the switch S_5 and regulator R_1 would be placed in `mySoC_PD` and the switch S_6 moved up to `core_PD`. Either methodology is valid and the supply networks are equivalent.

UPF provides the flexibility to specify the network in whatever way makes sense for a given design or methodology, including what can be switched. In this example, only the power supplies are switched; however, switches can also be placed on grounds, bias nets, and other named supply functions. This provides complete flexibility in how the supply distribution is controlled.

A UPF supply network defines a directed acyclic graph (DAG) when inout supply ports are not used. Limiting the supply network to a DAG representation simplifies the evaluation of the supply network state. This simplification enables efficient verification of the logic design and supply network within a digital simulation environment.

The state of the supply network is simply the state of each control element (switch) in the network and the values of any root supply driver at any given point in time. Absolute relationships are easy to infer from the supply network specification (e.g., `core_PD` is always on when `sub_PD` and `ana_PD` are on). To capture situations that are never intended to occur, UPF can also define legal and illegal power states (see [Clause 4](#)). The named power states and the legality of a named or unnamed power state can be used to optimize the design implementation and to facilitate design verification.

1.5 Extending logic specification

UPF defines extensions of the logic design with power-specific capabilities and constraints without modifying the original logic specification. UPF provides designers with the guarantee they intuitively expect, i.e., by adding a UPF power specification to a logic design or by changing or replacing an existing UPF specification for a logic design, designers do not need to touch the original logic specification or re-verify the logic functionality independent of the power specification. The UPF standard also facilitates the reuse of the logic specification in contexts where an explicit power-design intent is not required or where the power-design intent can change from one implementation of the design to another. [Figure 3](#) illustrates how UPF extends the logic design by adding power functionality while leaving the original functionality unmodified.

[Figure 3](#) also demonstrates the addition of retention and isolation functionality to the logic design. Isolation is required to ensure undefined outputs from powered-down design elements do not drain power from those design elements that are not powered down. Isolation also ensures a specific logic value is driven from the power domain's outputs. Assuming the output CNT in [Figure 3](#) can serve as an interrupt or reset indicator to another logic element, if the isolation of CNT did not take into account the active level of the interrupt or reset interpretation of its value, then powering down the domain containing this counter could result in undesired side-effects, e.g., logic in another power domain entering and remaining in a reset or interrupt-handling mode. UPF supports the specification of isolation strategies that provide information on the clamp values and location of the isolation logic.

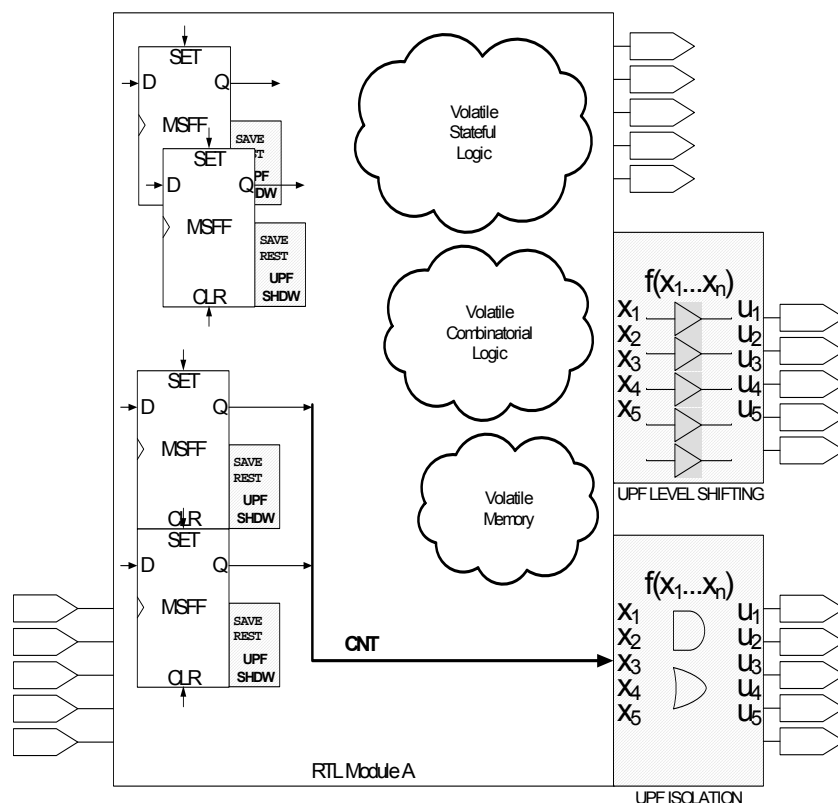


Figure 3—UPF extends the logic design

Retention is the ability to save the value of a sequential element in a power domain prior to switching off the power to that element and then later restoring its value after power has been enabled for the element. Behavioral and implementation retention semantics can be specified for sequential elements that require save and restore capabilities.

Isolation and retention both demonstrate UPF's concise specification characteristics and its flexibility in specification. A general strategy can be defined for both isolation and retention. This strategy is then applied to all ports or sequential elements governed by the scope of the strategy. Specific overriding of a general strategy allows for the management of exceptional situations. Flexibility is realized through the recognition that isolation and retention behavior, for any given design or implementation, may be more complex or require a different connectivity than the predefined general behaviors supported by UPF. In these situations, mapping of the isolation or retention to specific verification (functional behavior) and implementation models is supported.

1.6 Conventions used

Each clause that details any UPF commands defines its own conventions and meta-syntax as needed (see also [Clause 6](#) through [Clause 8](#)).

1.7 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 underlined-blue text (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 layered are shown in **boldface-green text**. See also [6.4](#).

1.8 Contents of this standard

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

- [Clause 2](#) provides references to other applicable standards that are presumed or required for this standard.
- [Clause 3](#) defines terms and acronyms used throughout the different specifications contained in this standard.
- [Clause 4](#) describes power domains and specifies the conventions for naming objects, their name space resolution, and conflict resolution.
- [Clause 5](#) defines simulation semantics for various UPF commands.
- [Clause 6](#) details the syntax and semantics for each UPF command.
- [Clause 7](#) details the syntax and semantics for each UPF query and the **find_objects** command.
- [Clause 8](#) describes the syntax and semantics of the Switching Activity Interchange Format (SAIF).
- Annexes. Following [Clause 8](#) are a series of annexes.

NOTE—The normative portion of this document is organized in two major sections. The first, [Clause 1](#) through [Clause 7](#), defines the UPF capabilities for capturing low power design intent. The second, [Clause 8](#), defines a format for capturing switching activity information, which can be used for analysis and estimating power consumption. While designing electronics to minimize power would benefit from tools supporting both sets of capabilities, the two sections hold no dependencies between one another.¹

¹Notes in text, tables, and figures are given for information only and do not contain requirements needed to implement the standard.

2. Normative references

The following referenced documents are indispensable for the application of this document (i.e., they must be understood and used, so each referenced document is cited in 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/IEEE 61691-1-1, Behavioural languages—Part 1: VHDL language reference manual.^{2, 3}

IEEE Std 1800™, IEEE Standard for SystemVerilog Unified Hardware Design, Specification, and Verification Language.⁴

3. Definitions, acronyms, and abbreviations

For the purposes of this document, the following terms and definitions apply. *The Authoritative Dictionary of IEEE Standards Terms* [B1]⁵ should be referenced for terms not defined in this clause.

3.1 Definitions

3.1.1 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.

3.1.2 active scope: The **instance** specified by the **set_scope** command (see 6.50).

3.1.3 activity: Any change of the inputs of an **element**, regardless of whether there are any changes on the output.

3.1.4 ancestor: Any **instance** between the **active scope** in the **logic hierarchy** and its root, including the root. However, when the **active scope** is the root, it does not have any ancestors.

3.1.5 automatic supply net connection: A supply net connection created as a result of the association of a **supply net** of a **supply set** identified by the function the net performs for the set and the **pg_type** attribute on a **supply port**. *See also* **explicit supply net connection** and **implicit supply set connection**.

3.1.6 command: A Tcl procedure [B5] defined to specify UPF power design intent with one or more **design objects** as its target.

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

3.1.8 connection: The attachment of a **port** to a **net**. *See also* **HighConn** and **LowConn**.

²IEC publications are available from the Sales Department of the International Electrotechnical Commission, Case Postale 131, 3, rue de Varembe, CH-1211, Genève 20, Switzerland/Suisse (<http://www.iec.ch/>). IEC publications are also available in the United States from the Sales Department, American National Standards Institute, 25 West 43rd Street, 4th Floor, New York, NY 10036, USA (<http://www.ansi.org/>).

³IEEE publications are available from the Institute of Electrical and Electronics Engineers, Inc., 445 Hoes Lane, Piscataway, NJ 08854, USA (<http://standards.ieee.org/>).

⁴The IEEE standards or products referred to in this clause are trademarks of the Institute of Electrical and Electronics Engineers, Inc.

⁵The number in brackets correspond to those of the bibliography in [Annex A](#).

3.1.9 corruption semantics: The rules defining the behavior of the logic in the context of the supply network specification, which might remove support of normal logical operation from time to time, as distinguished from the default always-on logical behavior.

3.1.10 design element: An **instance** of a SystemVerilog module (see IEEE Std 1800),⁶ VHDL entity (see IEC/IEEE 61691-1-1), or library cell. The term design element is often abbreviated to **element**.

3.1.11 design object: A design object is any object that can be declared or created in the **logic hierarchy**. Design objects include (but are not limited to) wires, registers, switches, **ports**, **supply nets**, and **design elements**. The term design object is often abbreviated to **object**.

3.1.12 driver: Any **design element** that supplies a value to a **net**, either directly, or indirectly via a **port**. Ports are not drivers.

3.1.13 explicit supply net connection: A supply net connection that is explicitly specified in a **connect_supply_net** command (see [6.13](#)). *See also automatic supply net connection and implicit supply net connection.*

3.1.14 extent: The set of **design elements** that comprise a **power domain**.

3.1.15 HighConn: This indicates the hierarchically higher (closer to the root) port connection.

3.1.16 hierarchical separator character: A special character used in composing hierarchical port, pin, net, or instance names from simple identifiers.

3.1.17 implicit supply set connection: The **supply nets** of a **supply set** are implicitly connected to those elements in the extent of a power domain, or those elements targeted by an isolation, retention, or level-shifter strategy, when no **supply port** exists on the element. *See also automatic supply net connection and explicit supply net connection.*

3.1.18 instance: A particular occurrence of a **design object**.

3.1.19 isolation: Techniques used to provide defined behavior of logic signals associated with independently switched supplies.

3.1.20 isolation cell: A **design element** that passes logic values during normal mode operation and clamps its output to some specified logic value when the control signal is asserted.

3.1.21 level-shifter: A **design element** that translates signal values from an input voltage swing to a different output voltage swing.

3.1.22 logic design: Design functionality defined using an HDL, such as SystemVerilog or VHDL.

3.1.23 logic hierarchy: The tree structure of **design elements** specified within a **logic design**.

3.1.24 LowConn: This indicates the lower (further from the root) port connection.

3.1.25 map: Substitute a specific **model** for an abstract behavior.

3.1.26 merged domain: A single **power domain** that is the union of the elements that were members of the power domains being merged. The domains that are merged into the merged domain are no longer visible.

⁶For information on references, see [Clause 2](#).

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

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

3.1.29 net: The collection of interconnections between a collection of ports. A net may be named or anonymous. At a module boundary, a named net may be considered as having two parts, the **HighConn** and **LowConn** portion of the net relative to a particular module or cell boundary.

3.1.30 parent: The immediate **ancestor** of a given **instance**.

3.1.31 pg_type: Supply ports on components can be labeled with a “Power/Ground Type” name; e.g., the Liberty definition [B6] of **pg_type**.

3.1.32 port: A **connection** on the interface of a SystemVerilog module or VHDL entity.

3.1.33 power domain: A collection of **design elements** that share a **primary supply set**. A power domain may also have additional supplies, including **retention** and **isolation** supplies.

3.1.34 power state: The state of a **supply net**, **supply port**, **supply set**, or **power domain**. The power state of a **supply net** or **supply port** is the **state** and **voltage** values of that **supply net** or **supply port**. The power state of a **supply set** is determined by the state of the **supply nets** in the set or by a logic expression if the supply network has not yet been created. The power state of a **power domain** is determined by the state of **supply sets** associated with the domain and **supply sets** or **supply nets** referenced in a named power state’s **-supply_expr** defined for the domain. The term power state is often abbreviated to **state**.

3.1.35 power state table: A table that captures the legal combinations of **power states** for a set of **supply nets**.

3.1.36 primary supply set: The **supply net** connections inferred for all elements in the **power domain**, unless overridden elsewhere.

3.1.37 rail: See the Liberty definition [B6] of **power_rail**.

3.1.38 receiver: Any **design element** that uses the value of a **net** as an input, either directly, or indirectly via a **port**. Ports are not receivers.

3.1.39 regular expressions: See the Tcl definition [B4] of *regular expressions*.

3.1.40 regulator: A **design element** that takes a set of input **supply nets** and provides the source for a set of output **supply nets**. The output voltage is a function of combining the input voltage and the logical state of any control signals.

3.1.41 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.

3.1.42 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 following the application of a suitable save and restore protocol sequence.

3.1.43 rooted name: The name of a **design object** in the active scope or one of its descendants. It may include a **hierarchical separator character**, but may not start with the **hierarchical separator character**.

3.1.44 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**.

3.1.45 scope: A particular **design element** in the **logic hierarchy**.

3.1.46 sequential element: An **object** containing latch or flip-flop functionality.

3.1.47 simple name: A name in the active scope. The name does not contain any **hierarchical separator characters**.

3.1.48 simstate: The level of operational capability supported by the active state of a **supply set**.

3.1.49 source: The active device or port connection that propagates a value on a **net**.

3.1.50 state element: A storage device such as a flip-flop, latch, memory element, or stored value that would infer a **sequential element** from HDL code.

3.1.51 subdomain: A member of the set of subdomains comprising a composite power domain; each subdomain retains its original identity as well as the collective identity of the composite power domain (see [6.15](#)).

3.1.52 supply function: The purpose of an individual **net** in a **supply set** (e.g., power, ground, etc.)

3.1.53 supply net: A **net** with **power state** semantics.

3.1.54 supply port: A **port** with **power state** semantics.

3.1.55 supply set: A collection of **supply nets** that provide a power source.

3.1.56 supply source: A supply port that propagates but does not originate a supply state and voltage value.

3.1.57 switch: A **design element** that conditionally connects one or more input **supply nets** to a single output **supply net** according to the logical state of one or more control inputs.

3.1.58 unnamed power state: A power state that is not predefined nor defined by the **add_power_state** command (see [6.8](#)). *See also* **named power state**.

3.1.59 UPF simulation: A simulation run that depends on semantics described in this standard.

3.2 Acronyms and abbreviations

DAG	directed acyclic graph
EDA	electronic design automation
HDL	hardware description language
IP	intellectual property
PST	power state table
RTL	register transfer level

SAIF Switching Activity Interchange Format

UPF Unified Power Format

VCT value conversion table

VHDL VHSIC hardware description language

4. Power domains, supply sets, name spaces, and precedence

This clause provides an overview of power domains, supply sets and names in UPF.

4.1 Power domains

From a top-down view of power design specification, the fundamental object is the power domain. A *power domain* is a collection of design elements. Unless otherwise specified, elements of a power domain share a common primary supply set (see 4.3). The association of a primary supply set to all elements in a domain provides the fundamental semantics and automation opportunities in UPF as the primary supply set is implicitly connected to all elements within the domain.

The design consists of a hierarchical tree of design elements (*logic hierarchy*). The logic hierarchy level where a power domain is created is called the *scope* of the power domain. The set of design elements that belong to a power domain are said to be the *extent* of that power domain. This distinction is important—while a design element can be the scope of multiple power domains, it can be in the extent of one and only one power domain. It shall be an error if any design element is not included in a power domain and does not have supply nets connected to all supply ports of all elements after the application of UPF commands.

Each power domain exists within a scope of the logic hierarchy. A design element is a member of the power domain that includes the design element's parent instance, unless the design element has been explicitly included as an element of another power domain whose scope is the element or an ancestor of the element.

In addition to creating the power domains, UPF commands create and connect new objects that did not exist in the HDL description—e.g., switches, supply nets, supply ports, isolation elements, and level shifting elements. UPF provides the ability to specify the low power design intent. By tightly associating the power domain and other UPF created objects and connections with the HDL design, designers can design, verify, and implement complex chips complete with all power-related connections and functionality.

- For verification, every object in the power domain and its power distribution and control network exists within the logic design. This allows the designer to see these objects within the context of the design.
- For implementation, UPF provides a convenient way to manipulate and report information on groups of cells that share common power, ground, and bias supply connections.
- UPF ensures that the implementation semantics have matching simulation semantics; the results of the semantic interpretation of UPF are consistent across simulation and implementation.

A power domain can be either contiguous or non-contiguous. The power domain is *contiguous* if a connection from any object in the extent of the power domain to any other object in the extent does not require leaving that power domain; otherwise the domain is *non-contiguous*. All elements within the extent of the domain shall be within the scope of the domain or its descendants.

A net that is not driven by design elements in a particular domain is not corrupted by that domain; input ports and assignment statements are not considered drivers in this context.

Supply network objects (supply nets, supply ports, supply sets, and switches) are created within the logic hierarchy independent of the power domain definitions. This allows sharing of common components of the supply distribution network across multiple power domains independent of the power domain the object is within. Logic nets and ports created in UPF are created within the logic hierarchy independent of the power domain definitions. This allows the power control network to be created and distributed across power domains. All other UPF objects are created within the context of a power domain.

4.2 Supply nets and ports

Supply nets transport an electrical current. Supply ports provide the ability to connect a supply net to a design element (including switches). Supply nets and ports may be created in UPF or in the HDL design. If created in the HDL, the net or port shall be of the supply net type defined in the UPF SystemVerilog or VHDL package (see [Annex B](#)). When created in UPF, the supply net is created within the scope of a design element of the logic hierarchy. When created in UPF, a supply port is created on the interface of an element in the logic hierarchy. The name of the supply net or port cannot conflict with the name of an existing net or port (logic or supply) defined for that element.

Supply ports provide the ability for a design element to use a supply net that exists external to it. Supply ports consist of two halves. The first half is the `HighConn` side, which is visible to the parent of the design element whose interface contains the port. The second half is the `LowConn` side, which is visible internal to the design element whose interface contains the port.

When a supply or logic net in the active scope is connected to a supply or logic port on a child instance, the connection is made to the `HighConn` side of the port. When a supply or logic net in the active scope is connected to a port defined on the interface of the design element that is the active scope, the connection is made to the `LowConn` side of the port. If there is no net connected to a port referenced by a UPF command in a context requiring a net name, a net with the same name as the port is implicitly created in the active scope and implicitly connected to the port. If a port is referenced by a UPF command in a context requiring a net, an existing net that is already connected to the port shall be used. However, if no net has been connected to the port, a net with the same name as the port is implicitly created in the active scope and implicitly connected to the port.

A supply or logic net in the active scope can be connected to a port only if the port is directly visible in the same scope as the net or within the descendent tree of that scope. If the port is not directly visible in the same scope as the net, additional ports and nets shall be implicitly created to establish the connection from the net to the port. The 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. See [6.35](#).

NOTE—Nets are propagated as necessary through the descendant tree 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.

The electric current transported by a supply net is originated by a root supply driver. A *root supply driver* is the origin of a supply, e.g., an on-system voltage regulator, bias generator modeled in HDL, or an off-chip supply source. A root supply driver may be conditionally propagated by a switch (modeled in HDL or created in UPF, see [6.20](#)). 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 (see [6.22.1](#)).

4.3 Supply sets

Because a single supply net by itself has no meaning relative to the power being supplied to any design element, UPF provides the ability to create supply sets. UPF predefines the following supply set handles for a domain: **primary**, **default_retention**, and **default_isolation**.

A *supply set* relates multiple supply nets as a complete power source for one or more design elements.

Each supply net in a supply set provides a function. UPF predefines the following supply net functions: **power**, **ground**, **pwell**, **nwell**, **deeppwell**, and **deepnwell**. A pre-defined supply set may be referenced through a supply set handle. Additional supply net functions may also be defined for a predefined supply set.

Once created, a supply set can be associated with a domain, retention strategy, or isolation strategy for a specific purpose. The supply nets of a set are implicitly “routed” to all elements that require them. Implicit “routing” includes the implicit creation of supply ports and nets required to propagate the supply net to each element where it is required.

UPF defines implicit and automatic connection semantics for the supply nets of a supply set based on the purpose of the supply set in a given domain or strategy context and the function that a supply net performs in the context of the supply set.

NOTE—A supply net may be included in more than one supply set. The function the supply net performs in one supply set is unrelated to the function it may perform in any other supply set in which it is included.

Supply nets can be connected in one of the following ways: explicitly, automatically, or implicitly.

4.3.1 Explicit connection of supply nets

A supply net may be *explicitly* connected to a port via a **connect_supply_net** command (see 6.13).

4.3.2 Automatic connection of supply nets

There are three ways supply nets can be automatically connected. They are as follows:

- a) Supply nets are *automatically* connected to supply ports of design elements in a power domain based on *pg_type* specification (see 6.10).
- b) Individual supply nets may be automatically connected to a port using a **connect_supply_net** command (see 6.13) with *pg_type* attributes defined.
- c) The supply nets of a supply set may be automatically connected to a port using a **connect_supply_set** command (see 6.14), where the *pg_type* attribute of the supply function for a net in the supply set is matched to a design element port with a corresponding *pg_type* attribute.

The supply nets of a supply set are automatically connected to the supply ports of elements under the following conditions:

- 1) The element has a port with a *pg_type* attribute (a string-typed HDL attribute named *pg_type* or an implementation library model with a *pg_type* attribute).
- AND
- 2) If the value of the *pg_type* attribute of the port matches a value specified via the **-connect** option of the **connect_supply_set** command (see 6.14), the supply net providing the function specified in the **-connect** option is automatically connected to the port.

The UPF specification is *erroneous* when it results in an incomplete set of connections for a design element.

The following precedence rules apply, from higher to lower precedence, for automatic and explicit connections of a supply net to a supply port.

connect_supply_net
connect_supply_set with **-elements** and **-connect**
connect_supply_set -connect

4.3.3 Implicit connection of supply nets

The supply nets of a supply set may be *implicitly* connected to design elements based on information in the power specification associating a supply set with a design element that is instantiated in the HDL code or implied by the power specification.

- Implicit connections provide the mechanism by which the supply nets of a supply set are connected to elements that do not have supply ports.

- The simstate simulation semantics for a supply set are applied to design elements implicitly connected to a supply set unless the **set_simstate_behavior** has been disabled (see 6.51).

The supply nets of a supply set are implicitly connected as the power source for an element when that element has *no supply ports*. Implicit connections define additional simulation semantic behavior (see Clause 5). Implicit connections are made under the following conditions:

- a) **Primary supply set**
The supply nets of a domain's primary supply set are implicitly connected to any design element from the logic hierarchy that is within the extent of the domain if the element has no supply ports defined on its interface.
- b) **Retention supply set**
The supply nets of a retention strategy's supply set are implicitly connected to the retention (i.e., balloon latch or shadow register) functionality that is implied for any register in the design to which the strategy applies.
- c) **Isolation supply set**
The supply nets of a supply set for an isolation strategy are implicitly connected to the corresponding isolation functionality implied by the application of the strategy.
- d) **Level-shifter supply sets**
The supply nets of a supply set for a level-shifting strategy are implicitly connected to the level-shifters implied by the application of the strategy.

4.3.4 Predefined supply set functions

The following names are reserved for predefined functions and may be used in any supply set definition:

- a) **power** is the supply net that provides the power function of the supply set and by default shall be connected to ports having the *pg_type* **primary_power**.
- b) **ground** is the supply net that provides the ground function of the supply set and by default shall be connected to ports having the *pg_type* **primary_ground**.
- c) **pwell** is the supply net that provides the pwell bias of the supply set and by default shall be connected to ports having the *pg_type* **pwell**.
- d) **nwell** is the supply net that provides the nwell bias of the supply set and by default shall be connected to ports having the *pg_type* **nwell**.
- e) **deeppwell** is the supply net that provides the deeppwell bias of the supply set and by default shall be connected to ports having the *pg_type* **deeppwell**.
- f) **deepnwell** is the supply net that provides the deepnwell bias of the supply set and by default shall be connected to ports having the *pg_type* **deepnwell**.

If a supply set is used within a domain, then a supply net shall be defined for each function required by that supply set for implementation. However, a simulator may support pre-implementation verification with only the specification that a supply set exists for the domain. A supply set that does not have supply nets defined for each of its required functions is *incompletely specified*. A reference to a supply net by its *symbolic* name is an *indirect reference*.

4.4 Naming rules

Names (identifiers) adhere to the following rules:

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

4.5 Name space semantics

Names also need to adhere to the following name space semantics:

- a) Objects created by a UPF command exist in the design; therefore, the names of those objects shall not conflict with a name that is visible within the same scope.
- b) 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 [6.35](#)) can be used to provide a template for some implicitly created objects (such as isolation). Supply nets may be implicitly created and connected to supply ports and logic nets may be implicitly created and connected to logic ports (see [4.2](#)).
- c) Many command arguments require the specification of object names, such as instance names, ports, registers, nets, etc. Unless otherwise specified or unambiguous from the context, a name reference can be a simple name or the hierarchical name of an object. The names are relative to the active UPF scope. If the name is a hierarchical name, it shall be in the descendant tree of the active scope. See also [6.50](#).
- d) Unless otherwise specified or unambiguous from the context, the terms *scope* and *UPF scope* are synonymous.
- e) 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.
- f) The following record field names are reserved in the specified context and cannot be redefined:
 - 1) Domain record field space
 - i) **primary**
 - ii) **default_retention**
 - iii) **default_isolation**
 - 2) Switch record field space
 - supply**
 - 3) Level-shifter strategy record field name space
 - i) **input_supply_set**
 - ii) **output_supply_set**
 - iii) **internal_supply_set**
 - 4) Isolation strategy record field name space
 - i) **isolation_supply_set**
 - ii) **isolation_signal**
 - 5) Retention strategy record field name space (see [6.33](#))
 - i) **retention_ref**
 - ii) **primary_ref**
 - iii) **save_signal**
 - iv) **restore_signal**
 - v) **UPF_GENERIC_CLOCK**
 - vi) **UPF_GENERIC_DATA**
 - vii) **UPF_GENERIC_ASYNC_LOAD**
 - viii) **UPF_GENERIC_OUTPUT**

- g) 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`.
- h) Name references include the refinements of complex objects such as: part-select or slice, bit-select, index-select, or field-select name (see IEEE Std 1800).

4.6 Attributes and HDLs

Hardware description languages 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 [Clause 6](#)). [Table 1](#) enumerates the HDL attributes defined for UPF-compliant implementations.

Table 1—Attribute and command correspondence^a

HDL attribute name	Attribute value specification	Equivalent UPF command arguments	See
UPF_clamp_value	<"0" "1" "Z" "latch" "any" "value">	set_isolation -clamp_value set_port_attributes -clamp_value	6.40 6.45
UPF_sink_off_clamp_value	<"0" "1" "Z" "latch" "any" "value">	set_isolation -sink_off_clamp_value set_port_attributes -sink_off_clamp_value	6.40 6.45
UPF_source_off_clamp_value	<"0" "1" "Z" "latch" "any" "value">	set_isolation -source_off_clamp_value set_port_attributes -source_off_clamp_value	6.40 6.45
UPF_pg_type	<i>pg_type_value</i> (see 4.3.4)	set_port_attributes -pg_type	6.45
UPF_related_power_pin	<i>port_name</i>	set_pin_related_supply -related_power_pin set_port_attributes -related_power_port	6.44 6.45
UPF_related_ground_pin	<i>port_name</i>	set_pin_related_supply -related_ground_pin set_port_attributes -related_ground_port	6.44 6.45
UPF_related_bias_pin	<i>port_name_list</i>	set_port_attributes -related_bias_ports	6.45
UPF_retention	<"required" "optional">	set_retention_elements -retention required set_retention_elements -retention optional	6.49
UPF_simstate_behavior	<"ENABLE" "DISABLE">	set_simstate_behavior	6.51
UPF_is_leaf_cell	<"TRUE" "FALSE">	set_design_attributes -is_leaf_cell	6.37

^aWhere two HDL attribute names are listed in the same table row, they are aliases for each other; where two UPF command arguments are listed in the same table row, they correspond to the same attribute.

It shall be an error if a UPF command conflicts with an attribute defined in an HDL file.

4.7 Precedence

To support concise, easily written low power specifications, UPF supports default and generic application of low power design intent. A UPF command specification can apply to a design object when specified directly on the object or an ancestor of the object. A command that applies to an element shall also apply to the descendents that match the constraints of the command. Application of multiple low power design intent specifications may result in overlapping UPF command effects.

When there is more than one power intent specification for a design object, the following precedence applies (ordered from highest to lowest priority):

- a) Direct UPF commands.
The power intent is applied through an explicit UPF command reference to a design object.
- b) Power intent applied to a parent is inherited by each child and transitively applied to descendents, except when a direct UPF command applies.
- c) Strategies specified with both the **-source** and **-sink** options.
- d) Strategies specified with a **-source** or **-sink**.

The precedence of a command is independent of the active scope during the command processing.

It shall be an error if the precedence rules fail to uniquely identify the power intent that applies to an object.

NOTE—When *list* arguments to command options are created using **find_objects** (see 7.1), 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**.

4.8 Lexical elements

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

Table 2—Special characters

Type	Character
logic hierarchy delimiter	/
escape character	\ (Only escapes the next character.)
bus delimiter, index operator, or within a <i>regex</i>	
range separator (for bus ranges)	:
record field delimiter	.

4.9 Units

The units for voltage values in UPF specifications shall be volts.

4.10 Boolean expressions

UPF expressions that are SystemVerilog Boolean expressions shall comply with the following restrictions:

- Control port names shall not conflict with any SystemVerilog keyword (see IEEE Std 1800).
- The Boolean expression syntax is limited to the tokens shown in [Table 3](#).

Table 3—Boolean expression syntax tokens

Token	Meaning
()	Parentheses
~ & ^	Bit-wise negation, AND, OR, XOR
! &&	Logical negation, AND, OR

5. Simulation semantics

This clause details the simulation semantics for the UPF commands (see also [Clause 6](#)).

5.1 Supply network creation

UPF supply network creation commands define the power supply network that connects power supplies to the design elements in a design. After these commands are applied, every design element 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 design elements. 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 [4.3](#)). The supply network defines how power sources are distributed to the design elements and how that distribution is controlled.

A supply port that *originates* a supply state and voltage value defines a *root supply driver*. Drivers may be created via the `create_root_supply_driver` function in the UPF packages (see [Annex B](#)). This allows models of regulators, switches, bias generators, etc. to be created in HDLs for connection to a supply network that may be defined in UPF.

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 not a root supply driver when the supply source can be traced through the switch to an input supply port (and the net connected to it) that is currently switched to the output port. 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 [6.22](#)).

NOTE—Since the supply net type is defined in the UPF packages, it is possible to create the supply network entirely in HDL source.

5.2 Supply network simulation semantics

5.2.1 Supply network initialization

Simulation initialization semantics are defined by the corresponding HDL. Existing models rely on the HDL initialization semantics for operations such as initializing ROMs, etc. Therefore, UPF does not change the existing initialization semantics of HDLs. To model design power-up behavior, it is recommended that verification environments allow the simulation to initialize according to normal HDL semantics (as though the design is powered-on prior to the simulation initialization sequence). The verification environment can, after initialization, power down the design and then take the design through the power-up sequence.

The initial state of supply ports and supply nets is **UNDETERMINED** with an unspecified voltage value. The state of a supply set is **DEFAULT_CORRUPT** when at least one of the functions defined for the supply set is not associated with a supply net. To modify these states, see [Annex B](#).

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 design element 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**.

5.2.2 Supply network update and 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 switches, but not all, are **FULL_ON** or any switch is **PARTIAL_ON** (see also [6.22.2](#)).

The state of a supply port and net is **FULL_ON** when it is connected to a root supply driver that is **FULL_ON**; the state is **UNDETERMINED** when there is no connected path to a root supply driver; the state is **OFF** when the root supply driver is **OFF**. By default, root supply drivers are **UNDETERMINED**; however, they may be turned **OFF**, **FULL_ON** or **PARTIAL_ON** to model the behavior and state of the root supply driver. The full/partial state represents the conductance of a switch along the supply path. The voltage value of a supply net is valid only when it is **FULL_ON** or **PARTIAL_ON**. A simulator may model the IR drop over the supply network. However, the semantics defined in this standard, such as the supply net resolution functions, presume an idealized supply network with no IR drop; the semantics for supply network resolution with modeled-IR drop are outside the scope of this standard.

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.

5.3 Power switch modeling

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 [6.20](#) 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 [6.20](#)):

```
create_power_switch kb
-output_supply_port {outp pda_vdd}
-input_supply_port {inp1 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)}}
-off_state {kb_off {(cp1 && cp2) || (!cp1 && !cp2)}}
-ack_port {ap yack {(cp1 ^ cp2)}}
```

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 inp1, 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 <= X;
        end
    endcase
```

```

        $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));

```

5.4 Power states

Supply nets, supply ports, supply sets, and power domains have power states. The definition of the power state of each object is different.

5.4.1 Power states of supply nets and ports

Supply nets and ports form the foundation of the power distribution network specification. Each supply port and net maintains two pieces of information: a supply state and a voltage value, which together constitute the power state of the supply net or port. The power state information for supply nets and ports is defined by the **supply_net_type**; the state of being **FULL_ON**, **OFF**, **PARTIAL_ON**, or **UNDETERMINED** and the voltage level (which is relevant only for the **FULL_ON** and **PARTIAL_ON** states). A supply net may be included in a supply set. Each supply set has a reference supply. The *default reference supply* for a supply set is an implicit reference supply, defined to be 0 volts. 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 0-volt implicit reference supply.

5.4.2 Power states of supply sets

Continuing up from the foundation of supply nets, the next step in defining the supply network is the supply set—a collection of supply nets that together define a complete power supply. The *supply set* is composed of two or more supply nets. Therefore, the power state for a supply set is specified in terms of the supply nets that compose the set. It is the combined states of the constituent supply nets that determine the following:

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

Supply set simulation semantics are applied to the elements connected to the supply set when the simstate behavior is enabled (see 6.51). The simulation behavior semantics (*simstate*) can be specified for a power state. The simstate specifies the level of operational capability supported by a supply set state. UPF defines several simstates that can be associated with supply set 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 power level of 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 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 changes or activity in the elements powered by the supply.
- b) **CORRUPT_ON_ACTIVITY**—The power level of the supply set is insufficient to support activity. However, the power level is sufficient that logic nets retain their state as long as there is no activity within the elements connected to the supply.
- c) **CORRUPT_STATE_ON_ACTIVITY**—The power level of the supply set is sufficient to support combinational logic, but it is not sufficient to support activity inside state elements, whether that activity would result in any state change or not.

- d) **CORRUPT_STATE_ON_CHANGE**—The power level of the supply set is sufficient to support combinational logic, but it is not sufficient to support a change of state for state elements.
- e) **NORMAL**—The power level of the supply set is sufficient to support full and complete operational (switching) capabilities with characterized timing.

The simstate specification provides digital-simulation tools sufficient information for approximating the power-related behavior of logic implicitly connected to the supply set with sufficient accuracy.

NOTE 1—When greater accuracy is desired or required, a mixed signal or full analog simulation shall 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.

5.4.2.1 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 *object_name* in an **add_power_state** command (see 6.8). A supply set is in the **DEFAULT_NORMAL** state when all supply nets of the set are **FULL_ON** (see Table 4).

Table 4—Default supply set state with simstate == NORMAL

simstate	power	ground	pwell	nwell	deeppwell	deepnwell	All others
NORMAL	FULL_ON, any voltage	FULL_ON, any voltage	FULL_ON, any voltage	FULL_ON, any voltage	FULL_ON, any voltage	FULL_ON, any voltage	FULL_ON, any voltage

The simstate for **DEFAULT_NORMAL** is **NORMAL**. The supply set is in the **DEFAULT_CORRUPT** power state when the states of its supply nets do not match any defined power state, including the **DEFAULT_NORMAL** predefined state, for the supply set. The simstate for **DEFAULT_CORRUPT** is **CORRUPT**.

5.4.2.2 Specification of min, nom, max voltage levels

Since the power state of a supply net or port is the active supply net value, specification of the min, nom, and max voltage levels cannot be made at the supply net- or port-level. The appropriate place to specify the min, nom, and max voltage levels is in a supply set's power state specification (see 6.8).

5.4.3 Power states of power domains

The power state of a domain is determined by the state of supply sets associated with the domain. For example, the definition of a domain's **MY_DOMAIN_IS_ON** power state would logically require that the primary supply set be in a power state that is a **NORMAL** simstate (all supply nets of the primary supply set are on and the current delivered by the power circuit sufficient to support normal operation. Similarly, a

SLEEP mode for the domain may require the primary supply set to be in power state whose simstate is **NOT_NORMAL**, perhaps **CORRUPT** state, while appropriate retention and isolation supplies are **NORMAL**.

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. Early in the design phase, it may not be known if the power or ground net (or both) of the primary supply will be switched.
 - 2) The logic signal(s) enabling isolation or triggering retention save or restore 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 inactive.
 - 4) The isolation enable(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 [6.8](#).

A domain's power state can be defined directly in terms of supply nets using **-supply_expr** in addition to the **-logic_expr**.

- If a domain's power state **-logic_expr** specification includes comparison of another domain's active state to a power state defined on that domain, it is equivalent to including the **-logic_expr** and **-supply_expr** specifications for that power state of the referenced domain in the definition of the power state.
- If a domain's power state **-logic_expr** specification includes comparison of a supply set's active state to a power state defined on that supply set, it is equivalent to including the **-logic_expr** and **-supply_expr** specifications for that power state of the referenced supply set in the definition of the power state.

5.4.3.1 Incompletely specified supply sets

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 might only be defined in terms of the state of logic elements. The state of the domain's supply sets is added later and is necessary for a complete supply network specification. 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** to incorporate supply network expressions (**-supply_expr**) or additional logic expressions (**-logic_expr**).

5.4.3.2 Simulation of power states

Each power domain and supply set is in a power state at any given point in time. A power state may be described and named via the **add_power_state** command (see [6.8](#)). These are the named power states.

Unnamed power states may be legal or illegal (see 6.8). The power state of a supply set is determined by the state of each of the supply nets in the set. The power state of a domain is determined by the state of each of the supply sets associated with the domain and any supply nets or supply sets directly or indirectly referenced in the **-supply_expr** for any named power state for the domain.

Implementations may allow the choice of an overall mode: **min**, **nom**, or **max**. If not specified otherwise, the default shall be the **nominal** mode. If **min** or **max** is specified, but only a single (nominal) value has been specified in the **-supply_expr** (see 6.8), that single value shall be used as the **min** or **max** value. The mode specification shall be used to determine if the **-supply_expr** of a power state evaluates *True*.

The power state of a supply set is determined as follows:

- a) After the signal network is updated, including the supply network (see 5.2.2) and prior to evaluation of the power state of power domains:
 - 1) If any named power state of the supply set has a non-empty **-supply_expr**, then
 - i) The **-supply_expr** for all named power states is evaluated.
 - ii) The supply set is in a named power state if the **-supply_expr** for that named power state evaluates *True*. Zero, one, or more named power states may match (its **-supply_expr** evaluates *True*). (The predefined **DEFAULT_NORMAL** and **DEFAULT_CORRUPT** power states are named power states.)
 - iii) In simulation, it shall be an error if any named power state matches and the matching state is defined as an illegal power state.
Implementation tools may presume illegal power states do not occur.
 - iv) In simulation, it shall be an error if the **-logic_expr** for any matching named power state does not evaluate *True*, see 5.4.3.3.
Implementation tools may presume **-logic_expr** evaluates *True* for any matching named power state.
 - v) In simulation, it shall be an error if the **-logic_expr** evaluates *True* for a non-matching power state (i.e., a power state for which **-supply_expr** does not evaluate *True*).
Implementation tools may presume **-supply_expr** evaluates *True* for any non-matching named power state.
 - 2) Otherwise, the supply set is in a named power state if its **-logic_expr** evaluates *True*. Zero, one or more named power states may match.
- b) The simstate semantics for all matching named power states are applied; the highest precedence corruption semantic prevails (see 5.4.2).
- c) If zero named power states match, the supply set is in an unnamed power state. The legality of the unnamed power state is determined as specified in 6.8.

The power state of a power domain is determined as follows:

- d) After the power state of supply sets is evaluated and prior to the evaluation of user-defined processes and **always** blocks, the power state of supply sets is evaluated.
 - 1) If any named power state of the domain has a non-empty **-supply_expr**, then
 - i) The **-supply_expr** for all named power states is evaluated.
 - ii) The domain is in a named power state if the **-supply_expr** for that named power state evaluates *True*. Zero, one, or more named power states may match.
 - iii) In simulation, it shall be an error if any named power state matches and the matching state is defined as an illegal power state.
Implementation tools may presume illegal power states do not occur.
 - iv) In simulation, it shall be an error if the **-logic_expr** for any matching named power state does not evaluate *True*, see 5.4.3.3.

Implementation tools may presume **-logic_expr** evaluates *True* for any matching named power state.

- v) In simulation, it shall be an error if the **-logic_expr** evaluates *True* for a non-matching power state (i.e., a power state for which **-supply_expr** does not evaluate *True*).

Implementation tools may presume **-supply_expr** evaluates *True* for any non-matching named power state.

- 2) Otherwise, the domain is in a named power state if its **-logic_expr** evaluates *True*. Zero, one, or more named power states may match.
- e) If zero named power states matches, the domain is in an unnamed power state. The legality of the unnamed power state is determined as specified in [6.8](#).

5.4.3.3 Evaluation of the power state **-supply_expr** and **-logic_expr** expressions and reporting of mismatches

As evaluation-ordering dependencies between the update of the power control network (e.g., logic nets controlling the sources of supply nets, such as switches and bias generators) and the update of the supply network are likely, the reporting of mismatches when one expression evaluates *True* and the other does not is performed in a manner that avoids false error reporting due to any simulation-ordering dependencies within the same simulation time step.

- If a power state's **-supply_expr** or **-logic_expr** expressions do not match (one evaluates *True* when the other does not) and an error is not already scheduled for that power state, an error shall be scheduled to be reported at the end of the simulation time step (after all other activity has been completed for the time step and prior to the simulator advancing the simulation time).
- If the **-supply_expr** or **-logic_expr** is evaluated due to changes in the nets or power states referenced in the expression and both expressions evaluate *True*, the scheduled error message, if any, shall be deleted.
- At the end of the simulation time step, if a power state **-supply_expr** and **-logic_expr** expression mismatch error is scheduled, that error shall be reported.

NOTE—In SystemVerilog, the end of the simulation time step when the power state error for a **-supply_expr** and **-logic_expr** mismatch occurs is called the *reactive region*.

5.4.3.4 Incomplete supply network verification semantics

For any function of a supply set that is not associated with a supply net, an implicit supply net is created and associated with the function. Implicitly created supply nets are initialized the same as explicitly created supply nets (see [5.2.1](#)).

Tools may provide mechanisms to change the power state of the supply set or power domain. Such mechanisms are outside the scope of this standard.

The power state of a supply set or power domain may also be changed from an HDL test bench in simulation using the `set_power_state` function defined in the UPF packages (see [Annex B](#)).

- a) When a supply set's power state is changed through `set_power_state` (directly or indirectly):
 - 1) An error is issued if the **-logic_expr** for the power state does not hold for the new state. There is no **-logic_expr** for the default supply set states: **DEFAULT_NORMAL** and **DEFAULT_CORRUPT**.
 - 2) Any supply nets referenced in the **-supply_expr** for the new state are forced into the state specified in the **-supply_expr** specification of the new state. Any state that satisfies the **-supply_expr** is permitted. The forcing of a supply port shall result in an event on the supply net. Therefore, any change in supply nets resulting from the execution of the `set_power_state` function shall be deferred until the start of the next supply network

evaluation cycle. Forcing of a supply port is equivalent to forcing any supply net connected to the output side of the port.

NOTE—Implicitly created supply nets may be referenced through symbolic names (e.g., `primary.power`).

- 3) The implicitly created supply nets of the set that are not referenced in the **-supply_expr** through their symbolic names (e.g., `primary.power`), shall have their state set as follows:
 - i) If the simstate of the state is **CORRUPT**: the state shall be set to **OFF** and the voltage value is unspecified.
 - ii) For any other simstate: the state shall be set to **FULL_ON** and the voltage value is unspecified.
- b) When a domain's power state is changed through **set_power_state** (directly or indirectly):
 - 1) It shall be an error if the **-logic_expr** for the power state does not hold for the new state. During this check, any sub-expressions that reference the power state of an object are assumed to hold.
 - 2) Any supply net referenced in the **-supply_expr** is forced as described in [a.2](#) for supply sets.
 - 3) For each domain or supply set referenced in **-logic_expr**, a recursive call to **set_power_state** shall be made to ensure the power states of those objects are set to the state required by the new state.

5.4.4 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 design specification, as well as an overall power design 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 all elements of a design shall be in a power domain, specification of power states for a system or subsystem are attributed to the power domain(s) that are defined for the (sub)system. 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 design element relative to the scope of another power domain.) Therefore, UPF allows the power state for a power domain to reference the power state of any power domain, supply set, supply net or supply port that is visible in the descendent tree of the scope of the 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.

5.5 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, a design element 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 6.8) is used to define the legal and illegal power states of power domains and supply sets. The `set_power_state` function in the UPF packages 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 `supply_net_type` in the UPF packages. The state of supply nets and ports can be set through the `assign_supply2supply` or `assign_supply_state` functions in the UPF packages. `assign_supply2supply` 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. `assign_supply_state` 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 4.10) 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
```

5.6 Simstate simulation semantics

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. The use of the term *elements* in the rest of this subclause refers only to the elements related to the supply set as described in this paragraph.

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 [6.51](#)).

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 VHDL `std_ulogic` and `std_logic` typed-objects 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**.

5.6.1 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.

5.6.2 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.

5.6.3 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.

5.6.4 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.

5.6.5 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.

5.6.6 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_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**.

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.

5.7 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**.

5.7.1 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.

5.7.2 Any state transition to CORRUPT_ON_ACTIVITY

In this case, the active 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.

5.7.3 Any state transition to CORRUPT_STATE_ON_CHANGE

In this case, the active 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).