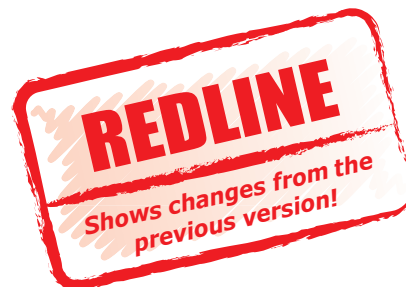


IEEE Standard for Design and Verification of Low-Power, Energy-Aware Electronic Systems

IEEE Computer Society

Sponsored by the
Design Automation Standards Committee



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

IEEE Std 1801™-2015
(Revision of
IEEE Std 1801-2013)

IEEE Standard for Design and Verification of Low-Power, Energy- Aware Electronic Systems

Sponsor

Design Automation Standards Committee
of the
IEEE Computer Society

Approved 5 December 2015

IEEE-SA Standards Board

This is a Redline Document produced by Techstreet, a business of Thomson Reuters.

This document is intended to provide users with an indication of changes from one edition to the next. It includes a full-text version of the new document, plus an indication of changes from the previous version.

Redlines are designed to save time and improve efficiencies by using the latest software technology to find and highlight document changes. More professionals are using valuable new technologies like redlines, to help improve outcomes in a fastpaced global business world.

Because it may not be technically possible to capture all changes accurately, it is recommended that users consult previous editions as appropriate. In all cases, only the current base version of this publication is to be considered the official document.

Redline Processing Notes:

1. ~~Red Text~~ - Red strikethrough text denotes deletions.
2. Blue Text - Blue underlined text denotes modifications and additions.

Grateful acknowledgment is made to the following for permission to use source material:

Accellera Systems Initiative

Unified Power Format (UPF) Standard, Version 1.0

Cadence Design Systems, Inc.

Library Cell Modeling Guide Using CPF

Hierarchical Power Intent Modeling Guide Using CPF

Silicon Integration Initiative, Inc.

Si2 Common Power Format Specification, Version 2.1

Abstract: A method is provided for specifying power intent for an electronic design, for use in verification of the structure and behavior of the design in the context of a given power-management architecture, and for driving implementation of that power-management architecture. The method supports incremental refinement of power-intent specifications required for IP-based design flows.

Keywords: bottom-up implementation, buffers, energy-aware design, IEEE 1801™, interface specification, IP reuse, isolation, level-shifting, power domains, power intent, power modeling, power states, successive refinement, supply states, repeaters, retention, Unified Power Format (UPF)

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

Copyright © 2016 by The Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 25 March 2016. Printed in the United States of America.

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

PDF: ISBN 978-1-5044-0105-0 STDGT20513
Print: ISBN 978-1-5044-0106-7 STDPD20513

IEEE prohibits discrimination, harassment, and bullying.

For more information, visit <http://www.ieee.org/web/aboutus/whatis/policies/p9-26.html>.

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.

Important Notices and Disclaimers Concerning IEEE Standards Documents

IEEE documents are made available for use subject to important notices and legal disclaimers. These notices and disclaimers, or a reference to this page, appear in all standards and may be found under the heading “Important Notice” or “Important Notices and Disclaimers Concerning IEEE Standards Documents.”

Notice and Disclaimer of Liability Concerning the Use of IEEE Standards Documents

IEEE Standards documents (standards, recommended practices, and guides), both full-use and trial-use, are developed within IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (“IEEE-SA”) Standards Board. IEEE (“the Institute”) develops its standards through a consensus development process, approved by the American National Standards Institute (“ANSI”), which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and participate without compensation from IEEE. While IEEE administers the process and establishes rules to promote fairness in the consensus development process, IEEE does not independently evaluate, test, or verify the accuracy of any of the information or the soundness of any judgments contained in its standards.

IEEE does not warrant or represent the accuracy or content of the material contained in its standards, and expressly disclaims all warranties (express, implied and statutory) not included in this or any other document relating to the standard, including, but not limited to, the warranties of: merchantability; fitness for a particular purpose; non-infringement; and quality, accuracy, effectiveness, currency, or completeness of material. In addition, IEEE disclaims any and all conditions relating to: results; and workmanlike effort. IEEE standards documents are supplied “AS IS” and “WITH ALL FAULTS.”

Use of an IEEE standard is wholly voluntary. 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.

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

IN NO EVENT SHALL IEEE BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO: PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE PUBLICATION, USE OF, OR RELIANCE UPON ANY STANDARD, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE AND REGARDLESS OF WHETHER SUCH DAMAGE WAS FORESEEABLE.

Translations

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

Official statements

A statement, written or oral, that is not processed in accordance with the IEEE-SA Standards Board Operations Manual shall not be considered or inferred to be the official position of IEEE or any of its committees and shall not be considered to be, or be relied upon as, a formal position of IEEE. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that his or her views should be considered the personal views of that individual rather than the formal position of IEEE.

Comments on standards

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

Comments on standards should be submitted to the following address:

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

Laws and regulations

Users of IEEE Standards documents should consult all applicable laws and regulations. Compliance with the provisions of any IEEE Standards document does not imply compliance to any applicable regulatory requirements. Implementers of the standard are responsible for observing or referring to the applicable regulatory 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

IEEE draft and approved standards are copyrighted by IEEE under U.S. and international copyright laws. They are made available by IEEE and are adopted 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 these documents available for use and adoption by public authorities and private users, IEEE does not waive any rights in copyright to the documents.

Photocopies

Subject to payment of the appropriate fee, IEEE will grant users a limited, non-exclusive license to photocopy portions of any individual standard for company or organizational internal use or individual, non-commercial use only. To arrange for payment of licensing fees, 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.

Updating of IEEE Standards documents

Users of IEEE Standards documents should be aware that these documents may be superseded at any time by the issuance of new editions or may be amended from time to time through the issuance of amendments, corrigenda, or errata. An official IEEE document at any point in time consists of the current edition of the document together with any amendments, corrigenda, or errata then in effect.

Every IEEE standard is subjected to review at least every ten years. When a document is more than ten years old and has not undergone a revision process, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE standard.

In order to determine whether a given document is the current edition and whether it has been amended through the issuance of amendments, corrigenda, or errata, visit the IEEE-SA Website at <http://ieeexplore.ieee.org/xpl/standards.jsp> or contact IEEE at the address listed previously. For more information about the IEEE-SA or IEEE's standards development process, visit the IEEE-SA Website at <http://standards.ieee.org>.

Errata

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

Patents

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

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

Participants

At the time this IEEE standard was completed, the P1801 Working Group had the following membership:

John Biggs, *Chair*
Erich Marschner, *Vice Chair*
Sushma Honnavara-Prasad, *Secretary*

Houssam Abbas	Anand Iyer	David Peterson
Paul Bailey	Fred Jen	Shreedhar Ramachandra
Guillaume Boillet	Tim Jordan	Judith Richardson
Conor Byrne	Sylvian Kaiser	Frederic Saint-Preux
Louis Cardillo	James Kehoe	Rich Scales
Shir-Shen Chang	Tim Kogel	Guido Schlothane
David Cheng	Rick Koster	Krishna Sekar
Cyril Chevalier	Shaji Kunjumohamed	Desinghu Pundi Srinivasan
Ashley Crawford	Kaowen Liu	Amit Srivastava
John Decker	Debajani Majhi	James Su
Stephan Diestelhorst	Ilija Materic	Haruyuki Tago
Shaun Durnan	Gene Matter	Ajay Thiriveedhi
Paul Floyd	Jon McDonald	Venki Venkatesh
Jerry Frenkil	Don Mills	Vita Vishnyakov
Alan Gibbons	Kevin Nesmith	Jon Worthington
Josefina Hobbs	Lawrence Neukom	Vojin Zivojnovic

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

Accellera Organization, Inc.	Intel Corporation	NVIDIA Corporation
Advanced Micro Devices	Japan Electronics and	PMC-Sierra, Inc.
(AMD)	Information Technology	Qualcomm, Inc.
ALDEC, Inc.	Industries Association (JEITA)	Silicon Integration Initiative,
ARM, Ltd.	Marvell Semiconductor, Inc.	Inc.
Broadcom Corporation	MediaTek, Inc.	STMicroelectronics
Cadence Design Systems, Inc.	Mentor Graphics	Synopsys, Inc.
Google	Micron Technology, Inc.	Verific Design Automation, Inc.
	Microsoft Corporation	

When the IEEE-SA Standards Board approved this standard on 8 December 2015, it had the following membership:

John D. Kulick, *Chair*
Jon Walter Rosdahl, *Vice Chair*
Richard H. Hulet, *Past Chair*
Konstantinos Karachalios, *Secretary*

Masayuki Ariyoshi	Joseph L. Koepfinger*	Stephen J. Shellhammer
Ted Burse	David J. Law	Adrian P. Stephens
Stephen Dukes	Hung Ling	Yatin Trivedi
Jean-Philippe Faure	Andrew Myles	Philip Winston
J. Travis Griffith	T. W. Olsen	Don Wright
Gary Hoffman	Glenn Parsons	Yu Yuan
Michael Janezic	Ronald C. Petersen	Daidi Zhong
	Annette D. Reilly	

*Member Emeritus

Introduction

This introduction is not part of IEEE Std 1801™-2015, IEEE Standard for Design and Verification of Low-Power, Energy-Aware Electronic Systems.

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

When the electronic design automation (EDA) industry began creating standards for use in specifying, simulating, and implementing functional specifications of digital electronic circuits in the 1980s, the primary design constraint was the transistor area necessary to implement the required functionality in the prevailing process technology at that time. Power considerations were simple and easily assumed for the design as power consumption was not a major consideration and most chips operated on a single voltage for all functionality. Therefore, hardware description languages (HDLs) such as VHDL (IEC 61691-1-1/IEEE Std 1076™^a) and SystemVerilog (IEEE Std 1800™^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 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, straightforward extensions to the methodology.

With process technologies below 90 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 have been 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 Unified Power Format (UPF) was developed to enable modeling of these new power-management techniques and to facilitate automation of design, verification, and implementation tools that must account for power-management aspects of a design. The initial version of UPF, developed by the Accellera Systems Initiative, focused primarily on modeling power distribution and its effects on the behavior of a system. In

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

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

May 2007 that initial version was donated to the IEEE, and in March 2009 a new version, IEEE Std 1801, was released. That update of UPF added many new features, including the concept of successive refinement, more abstract modeling of system-power states, and more abstract modeling of supply networks.

This document, the latest revision of IEEE Std 1801, makes available further enhancements to UPF, including enhanced concepts for modeling power states and transitions at all levels of aggregation, enhanced support for methodologies such as successive refinement and bottom-up implementation, and a detailed information model that serves as the basis for enhanced package UPF functions and query functions. This current version also provides support for component power modeling for system-level power analysis in virtual prototyping applications.

Contents

1. Overview	1
1.1 Scope	1
1.2 Purpose	1
1.3 Key characteristics of the Unified Power Format	1
1.4 Contents of this standard	3
2. Normative references	4
3. Definitions, acronyms, and abbreviations	4
3.1 Definitions	4
3.2 Acronyms and abbreviations	10
4. Concepts	11
4.1 Introduction	11
4.2 Design structure	11
4.3 Design representation	12
4.4 Power architecture	15
4.5 Power distribution	18
4.6 Power management	26
4.7 Supply states and power states	31
4.8 Simstates	38
4.9 Power intent specification	39
5. Language basics	45
5.1 UPF is Tcl	45
5.2 Conventions used	45
5.3 Lexical elements	47
5.4 Boolean expressions	51
5.5 Object declaration	53
5.6 Attributes of objects	53
5.7 Precedence	58
5.8 Generic UPF command semantics	60
5.9 effective_element_list semantics	60
5.10 Command refinement	63
5.11 Error handling	65
5.12 Units	65
5.13 SystemC language basic	65
6. Power intent commands	65
6.1 Introduction	65
6.2 Categories	65
6.3 add_parameter	66
6.4 add_port_state (legacy)	67
6.5 add_power_state	68
6.6 add_pst_state (legacy)	74
6.7 add_state_transition	76
6.8 add_supply_state	78
6.9 apply_power_model	78
6.10 associate_supply_set	80
6.11 begin_power_model	81
6.12 bind_checker	83

6.13 connect_logic_net.....	84
6.14 connect_supply_net.....	86
6.15 connect_supply_set.....	88
6.16 create_composite_domain.....	89
6.17 create_hdl2upf_vct.....	91
6.18 create_logic_net.....	92
6.19 create_logic_port.....	93
6.20 create_power_domain.....	94
6.21 create_power_state_group.....	97
6.22 create_power_switch.....	99
6.23 create_pst (legacy).....	106
6.24 create_supply_net.....	107
6.25 create_supply_port.....	111
6.26 create_supply_set.....	112
6.27 create_upf2hdl_vct.....	113
6.28 describe_state_transition (deprecated).....	114
6.29 end_power_model.....	114
6.30 find_objects.....	115
6.31 load_simstate_behavior.....	119
6.32 load_upf.....	120
6.33 load_upf_protected (deprecated).....	121
6.34 map_power_switch.....	121
6.35 map_repeater_cell.....	122
6.36 map_retention_cell.....	123
6.37 name_format.....	127
6.38 save_upf.....	128
6.39 set_correlated.....	129
6.40 set_design_attributes.....	130
6.41 set_design_top.....	131
6.42 set_domain_supply_net (legacy).....	132
6.43 set_equivalent.....	133
6.44 set_isolation.....	135
6.45 set_level_shifter.....	142
6.46 set_partial_on_translation.....	148
6.47 set_port_attributes.....	149
6.48 set_repeater.....	155
6.49 set_retention.....	158
6.50 set_retention_elements.....	162
6.51 set_scope.....	163
6.52 set_simstate_behavior.....	164
6.53 set_variation.....	165
6.54 upf_version.....	166
6.55 use_interface_cell.....	167
7. Power-management cell definition commands.....	169
7.1 Introduction.....	169
7.2 define_always_on_cell.....	170
7.3 define_diode_clamp.....	171
7.4 define_isolation_cell.....	172
7.5 define_level_shifter_cell.....	175
7.6 define_power_switch_cell.....	179
7.7 define_retention_cell.....	181
8. UPF processing.....	183
8.1 Overview.....	183
8.2 Data requirements.....	184

8.3 Processing phases	184
8.4 Error checking	188
9. Simulation semantics	188
9.1 Supply network creation	188
9.2 Supply network simulation	190
9.3 Power state simulation	191
9.4 Power state transition detection	194
9.5 Simstate simulation	194
9.6 Transitioning from one simstate state to another	197
9.7 Simulation of retention	198
9.8 Simulation of isolation	204
9.9 Simulation of level-shifting	205
9.10 Simulation of repeaters	205
10. UPF information model	205
10.1 Overview	205
10.2 Components of UPF information model	206
10.3 Identifiers in information model (IDs)	207
10.4 Classification of objects	210
10.5 Example of design hierarchy	217
10.6 Object definitions	218
11. Information model application programmable interface (API)	272
11.1 Tcl interface	272
11.2 HDL interface	282
Annex A (informative) Bibliography	344
Annex B (normative) Value conversion tables	345
B.1 VHDL_SL2UPF	345
B.2 UPF2VHDL_SL	345
B.3 VHDL_SL2UPF_GNDZERO	345
B.4 UPF_GNDZERO2VHDL_SL	346
B.5 SV_LOGIC2UPF	346
B.6 UPF2SV_LOGIC	346
B.7 SV_LOGIC2UPF_GNDZERO	346
B.8 UPF_GNDZERO2SV_LOGIC	346
B.9 VHDL_TIED_HI	347
B.10 SV_TIED_HI	347
B.11 VHDL_TIED_LO	347
B.12 SV_TIED_LO	347
Annex C (informative) UPF query examples	348
C.1 Utility procs	348
C.2 High-level procs	349
Annex D (informative) Replacing deprecated and legacy commands and options	352
D.1 Deprecated and legacy constructs	352
D.2 Recommendations for replacing deprecated and legacy constructs	353
Annex E (informative) Low-power design methodology	355
E.1 Simple System on Chip (SoC) example design	355
E.2 Design, verification, and implementation flow	358
E.3 Power intent of the example design	361

Annex F (informative) Power-management cell definitions in UPF and Liberty.....	382
F.1 Introduction	382
F.2 define_always_on_cell	382
F.3 define_diode_clamp	384
F.4 define_isolation_cell	385
F.5 define_level_shifter_cell	388
F.6 define_power_switch_cell	390
F.7 define_retention_cell	392
Annex G (informative) Power-management cell modeling examples.....	397
G.1 Modeling always-on cells	397
G.2 Modeling cells with internal diodes.....	403
G.3 Modeling isolation cells.....	405
G.4 Modeling level-shifters.....	422
G.5 Modeling power-switch cells.....	439
G.6 Modeling state retention cells	448
Annex H (informative) IP power modeling for system-level design.....	461
H.1 Overview of system-level IP power models	461
H.2 Content of system-level IP power models	462
H.3 Power calculation using power functions	463
H.4 Power model structure	465
H.5 Power model instantiation—example approach.....	466
Annex I (normative) Switching Activity Interchange Format	468
I.1 Syntactic conventions.....	469
I.2 Lexical conventions.....	470
I.3 Backward SAIF file.....	473
I.4 Library forward SAIF file	489
I.5 RTL forward SAIF file.....	497

IEEE Standard for Design and Verification of Low-Power, ~~Integrated~~ ~~Circuits~~ Energy-Aware Electronic Systems

IMPORTANT NOTICE: ~~This~~ IEEE Standards ~~is~~ documents are not intended to ensure safety, security, health, or environmental protection, ~~in all circumstances~~ or ensure against interference with or from other devices or networks. Implementers of ~~the~~ IEEE Standards documents are responsible for determining and complying with all appropriate safety, security, environmental, health, and interference protection practices ~~or~~ and all applicable laws and regulations ~~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~~ defines the syntax and semantics of a format used to ~~define the low-express~~ power intent in energy-aware electronic system design. *Power intent* includes the concepts and information required for specification and validation, implementation and verification, and modeling and analysis of ~~electronic systems and electronic intellectual property (IP)~~. The format ~~provides the ability to specify the supply network, switches, isolation, retention, and other aspects relevant to~~ power-managed ~~of an~~ electronic systems. ~~The~~ This standard also defines the relationship between the ~~low-power~~ intent captured in this format and design ~~specification and the logic design specification~~ intent captured via other formats (e.g., standard hardware description languages ~~(HDLs)~~ and cell libraries).

1.2 Purpose

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

1.3 Key characteristics of the Unified Power Format

The Unified Power Format (UPF) provides the ability for electronic systems to be designed with power as a key consideration early in the process. UPF accomplishes this by allowing the specification of what was traditionally physical implementation-based power information early in the design process—at the register transfer level (RTL) or earlier. Figure 1 shows UPF supporting the entire design flow. UPF provides a consistent format to specify power-design information that may not be easily specifiable in ~~an~~ [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 4 and 11.2). UPF also defines consistent semantics across verification and implementation, ~~i.e.,~~ [to check that](#) what is implemented is the same as what has been verified.

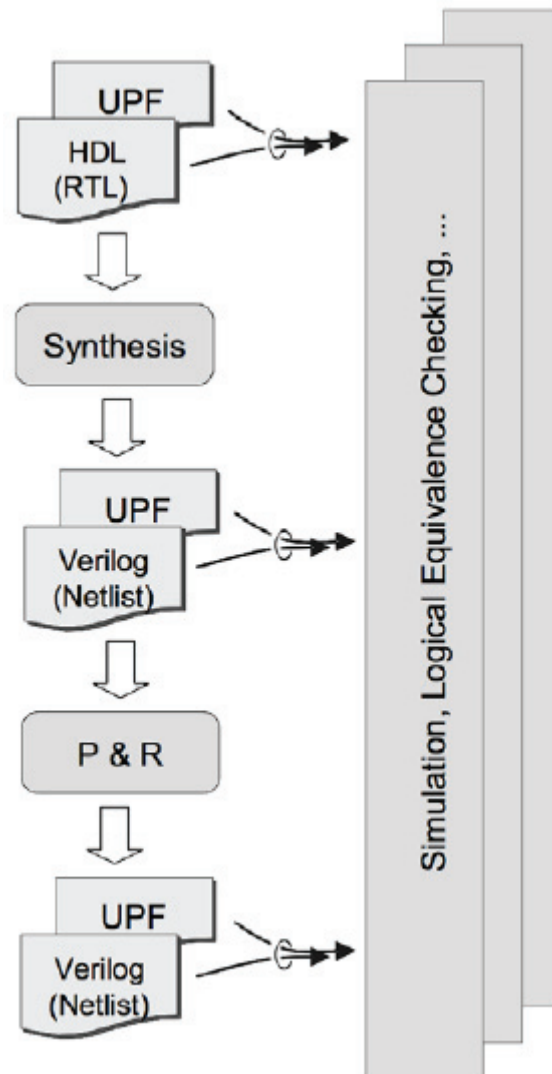


Figure 1—UPF tool flow

As indicated in Figure 1, UPF files are part of the design source and, when combined with the HDL, represent a complete design description: the HDL describing the logical intent and the UPF describing the power intent. Combined with the HDL, the UPF files are used to describe the intent of the designer. This collection of source files is the input to several tools, e.g., simulation tools, synthesis tools, and formal verification tools. UPF supports the successive refinement methodology (see 4.9) where power-intent information ~~will~~ grows along the design flow to provide needed information for each design stage.

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

UPF is a concise, power-intent specification capability. Power intent can be easily specified over many elements in the design. A UPF specification can be included with the other deliverables of [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 ~~(see Annex E)~~.

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

1.4 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 the basic concepts underlying UPF.
- Clause 5 describes the language basics for UPF and its commands.
- Clause 6 details the syntax and semantics for each UPF power intent command.
- Clause 7 details the syntax and semantics for each UPF power-management cell command.
- Clause 8 defines a reference model for UPF command processing.
- Clause 9 defines simulation semantics for various UPF commands.
- [Clause 10 defines the UPF information model.](#)
- [Clause 11 defines the UPF information model application programmable interface \(API\).](#)
- [Annex A lists potentially useful additional reference material.](#)
- [Annex B lists the predefined value conversion tables \(VCTs\) for use in power intent specifications.](#)
- [Annex C provides sample Tcl procs for retrieving power intent information.](#)

- [Annex D summarizes deprecated and legacy commands.](#)
- [Annex E provides an overview of UPF tool flows and use model with an illustrative example.](#)
- [Annex F provides a summary of UPF power-management cell command semantics and Liberty mappings.](#)
- [Annex G provides examples of UPF power-management cell modeling.](#)
- [Annex H provides an overview of UPF use model for system-level IP power modeling.](#)
- [Annex I defines the Switching Activity Interchange Format \(SAIF\) for representing power-related activity in a design.](#)
- ~~Annexes. Following Clause 9 are a series~~

2. Normative references

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

IEC 61691-1-1/IEEE Std 1076™, Behavioural languages—Part 1-1: VHDL Language Reference Manual.^{10, 11, 12}

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

[ISO/IEC 19501:2005, Information technology—Open Distributed Processing—Unified Modeling Language \(UML\) Version 1.4.2.](#)

¹⁰ [ISO/IEC](#) publications are available from the International Electrotechnical Commission (<http://www.iec.ch/>). IEC publications are also available in the United States from the American National Standards Institute (<http://www.ansi.org/>).

¹¹ IEEE publications are available from The Institute of Electrical and Electronics Engineers (<http://standards.ieee.org/>).

¹² The IEEE standards or products referred to in this clause are trademarks of The Institute of Electrical and Electronics Engineers, Inc.

3. Definitions, acronyms, and abbreviations

For the purposes of this document, the following terms and definitions apply. The *IEEE Standards Dictionary Online* [B1]¹³ should be consulted for terms not defined in this clause.¹⁴ Certain terms in this standard reflect their corresponding definitions in IEEE Std 1800TM or IEC 61691-1-1/IEEE Std 1076TM, or they are listed in Annex A.⁶

3.1 Definitions

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

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

¹³ The numbers in brackets correspond to those of the bibliography in Annex A.

¹⁴ *IEEE Standards Dictionary Online* subscription is available at:

http://www.ieee.org/portal/innovate/products/standard/standards_dictionary.html.

⁶~~Information on references can be found in Clause 2.~~

active power state: A power state whose logic expression ~~and, if present~~ —or, in certain cases, supply expression — evaluate to *True* at a given time.

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

analog port: A port that is part of a connection that delivers analog signals.

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

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

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

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

child domain (of a **HighConn** port on the **lower boundary** of a **power domain**): The power domain whose **upper boundary** contains the corresponding **LowConn** of the **HighConn port**.

coarse grain switch: A power switch that is used to generate switched supply for a group of library cells. This is identified using the attribute `switch_cell_type: coarse_grain` in Liberty and design attribute `UPF switch_cell_type coarse_grain` in Unified Power Format (UPF) (see Table 4).

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

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

configuration UPF: ~~The~~ [A Unified Power Format \(UPF\)](#) specification ~~for an intellectual property (IP) block that defines a particular~~ [of the power-management](#) configuration ~~of the block for use in a given system. The configuration UPF typically includes the~~ **constraint UPF** ~~and extends it with configuration specific details. Sometimes referred to as~~ **golden UPF**.

connected: Attached together via a direct connection.

constraint UPF: ~~The~~ [A Unified Power Format \(UPF\)](#) specification for an intellectual property (IP) block that defines constraints [for any instance of this IP block](#) that must be met by ~~any~~ [the power-management](#) configuration of the ~~IP block used in a larger~~ system. ~~Sometimes referred to as~~ **platinum UPF** [containing that instance](#).

correlated: [A pair of supply nets or a pair of supply sets that are deemed to be at the same point in their voltage range when being considered for level-shifting. As such, when voltage levels are considered between them they should be mutually compared, e.g., minimum to minimum and maximum to maximum.](#)

corruption semantics: The rules defining the behavior of logic [in](#) response to reduction or disconnection of power to that logic.

current scope: The design hierarchy location that serves as the immediate context for interpretation and execution of [Unified Power Format \(UPF\)](#) commands. Also, the **instance** specified by the **set_scope** command.

~~NOTE—See 6.52.~~

declared: Specified in the [hardware description language \(HDL\)](#) explicitly or implicitly via a [Unified Power Format \(UPF\)](#) command.

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

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

design hierarchy: A hierarchical structure of nested definitions described in ~~an~~ [a hardware description language \(HDL\)](#).

direct connection: A physical wire; or any [hardware description language \(HDL\)](#) construct(s) that synthesize(s) to a direct connection.

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

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

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

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

equivalent: A pair of **supply nets**, a pair of **supply sets** or [a pair of logic nets](#) that are considered to be interchangeable for certain purposes. *See also:* **electrically equivalent**; **functionally equivalent**.

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

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

fanout domain (of a given port to which a given strategy applies): The **power domain** containing any of the following: receiving logic for that **port**, or a **leaf-level cell** instance **HighConn** input **port** that is connected to the given **port**, or a design top module **LowConn** output **port** that is connected to the given **port**.

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

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

fine grain switch: A power switch that is used to generate switched supply for a single library cell. This is typically used to describe embedded macro power switches. This is identified using the attribute `switch cell type: fine grain` in Liberty and design attribute `UPF switch cell type fine grain` in Unified Power Format (UPF) (see Table 4).

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

generate block: In the [hardware description language](#) (HDL) code, this represents a level of design hierarchy, although a generate block is not itself an **instance**. After synthesis, generate blocks do not exist as an independent level of hierarchy. It is illegal to create any [Unified Power Format](#) (UPF) objects in a **scope** that corresponds to a generate block.

golden source: The design together with the constraint [Unified Power Format \(UPF\)](#) and the configuration UPF.

~~golden UPF: See: configuration UPF.~~

hard macro: A block that has been completely implemented and can be used as it is in other blocks. This ~~may~~ [can](#) be modeled by an [hardware description language \(HDL\)](#) module for verification or as a library cell for implementation.

hierarchical name: A series of names separated by the **hierarchical separator character**, the final name of which ~~may be any~~ [is a](#) legal [hardware description language \(HDL\)](#) name or [Unified Power Format \(UPF\)](#) name, and each preceding name is the name of an **instance** or **generate block** in which the following name is **declared**. *See also:* **hierarchical separator character**.

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

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

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

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

instance: A particular occurrence of a SystemVerilog module (see IEEE Std 1800), [very high speed integrated circuit \(VHSIC\) hardware description language \(VHDL\)](#) entity (see IEC 61691-1-1/IEEE Std 1076), or library cell at a specific location within the design hierarchy.

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

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

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

leaf-level cell: An instance that has no descendants, or an instance that ~~has the attribute UPF_is_leaf_cell associated with it~~ [is a soft or hard macro](#).

~~NOTE—See Table 4.~~

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

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

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

logically equivalent: Logic ports/nets that are directly connected without any intervening logic and therefore have the same value at all times from the perspective of any sink.

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

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

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

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

model: A SystemVerilog module, [very high speed integrated circuit \(VHSIC\) hardware description language \(VHDL\)](#) entity/architecture, or Liberty cell.

named power state: A power state defined using `add_power_state`, ~~`add_port_state`~~, or ~~`add_pst_state`~~ for a supply set, ~~or~~ power domain, [composite domain](#), [group](#), [model](#), or [instance](#), including the ~~`DEFAULT_NORMAL`~~ and ~~`DEFAULT_CORRUPT`~~ power states [ON](#) and [OFF](#) that are predefined for [supply sets](#) and the [power states UNDEFINED](#) and [ERROR](#) that are predefined for [all objects that have power states](#).

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

net segment: A direct connection within a single **instance**.

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

[parent domain \(of a LowConn port on the upper boundary of a power domain\):](#) The [power domain](#) whose [lower boundary](#) contains the corresponding [HighConn](#) of the [LowConn](#) port.

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

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

~~**platinum UPF:** See: **constraint UPF**.~~

port: A **connection** on the interface of a SystemVerilog module or [very high speed integrated circuit \(VHSIC\) hardware description language \(VHDL\)](#) entity. Also, a **port** on the **interface of a power domain**.

power domain: A collection of **instances** that are treated as a group for power-management purposes. The **instances** of a power domain typically, but do not always, share a **primary supply set**. A power domain ~~may~~ [can](#) also have additional supplies, including **retention** and **isolation** supplies.

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

power state: ~~The state~~ [A subset of a the functional states of an object that have the same characteristics with respect to power supply net, supply port, \(for a supply set,\) or power consumption \(for a power domain, composite domain, group, model, or instance\).](#)

~~NOTE—See Clause 4.~~

power state table (PST): A table that ~~captures~~ [specifies](#) the legal combinations of ~~power~~ [supply](#) states for a set of supply [objects \(supply ports, supply nets, and/or supply nets set functions\).](#)

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

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

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

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

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

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

rooted name: The **hierarchical name**, relative to the **current scope**, of an object in the **logic hierarchy** or a [Unified Power Format \(UPF\)](#) object defined for a **scope** in the hierarchy.

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

root supply driver: The origin of a supply, e.g., an on-system voltage regulator, bias generator modeled in [hardware description language \(HDL\)](#), or an off-chip supply source; [also, any supply object that functions as a root supply driver, including a primary supply input to the design, a leaf-level instance supply output port, a power switch output port, and any supply object that is an input to a resolved supply net.](#) See also: **supply source**.

self domain (of a **port** to which a given strategy applies): [The power domain for which the strategy has been defined.](#)

scope: [A region in which names may be defined; such a region is either an hardware description language \(HDL\) model or instance in the logic hierarchy or a Unified Power Format–defined global context, power state table, supply set, power domain, composite domain, group, or strategy.](#)

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

~~NOTE—See 6.7 and 6.11.~~

silicon UPF: See: **implementation UPF** [\(Unified Power Format\)](#).

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

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

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

soft macro: An **instance** that is represented by the original register transfer level (RTL) and Unified Power Format (UPF) from which its implementation is (or will be) derived. Additionally, **ancestor power intent objects** are not available for use within the **scope** of the **instance**.

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

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

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

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

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

supply net: An [hardware description language \(HDL\)](#) representation of a **power rail**.

supply port: A connection point for **supply nets**.

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

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

supply subnet: A set of electrically equivalent supply ports, supply nets, and/or supply set functions.

switch: An **instance** 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.

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

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

3.2 Acronyms and abbreviations

CMOS complementary metal oxide semiconductor

DFT Design for Test

EDA	electronic design automation
HDL	hardware description language
IP	intellectual property
MSFF	master-slave flip-flop
NMOS	N-channel metal oxide semiconductor
PG	power/ground
PMOS	P-channel metal oxide semiconductor
PST	power state table
ROM	Read-only memory
RTL	register transfer level
SAIF	Switching Activity Interchange Format
SoC	System on Chip

Tcl Tool Command Language

UPF Unified Power Format

VCT value conversion table

VHDL [very high speed integrated circuit](#) (VHSIC) hardware description language

VHSIC Very High Speed Integrated Circuit

4. ~~UPF~~ Concepts

4.1 Introduction

~~This~~ Clause 4 provides an overview of concepts involved in defining power intent using [Unified Power Format](#) (UPF). These concepts include those related to the representation of the design structure and functionality in one or more hardware description languages (HDLs), as well as those related to power-management structures and functionality defined for and/or added to the design to model intended power-management capabilities.

The structure and functionality of a design is specified using HDLs such as Verilog~~[B2]~~, SystemVerilog, or [very high speed integrated circuit \(VHSIC\) hardware description language \(VHDL\)](#). Each HDL ~~may have~~ [has](#) specific terminology and concepts that are unique to that language, but all HDLs share some common concepts and capabilities. A typical design may be expressed in one or more HDLs.

UPF is defined in terms of a generalized abstraction of an HDL-based design hierarchy. This abstraction enables the UPF definition to apply to a design expressed in any of the three HDLs previously mentioned, or in any combination thereof, while at the same time minimizing the complexity of the UPF definition. ~~This~~ Clause [4](#) presents the abstract model and maps it to specific HDL concepts.

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

4.2 Design structure

4.2.1 Transistors

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

4.2.2 Standard cells

Transistors are seldom modeled individually in an HDL description; typically, collections of transistors are represented by standard cells that have been developed as part of a particular technology library, which is usually expressed in the Liberty library format (see [B4]). Such cells typically have a primary supply (power and ground) and ~~may~~ [can](#) also have a secondary supply for related behavior (e.g., state retention).

4.2.3 Hard macros

A library ~~may~~ [can](#) also contain hard macros, which provide predefined physical implementations for much larger and more complex functions. A hard macro ~~may~~ [can](#) have multiple supplies.

4.3 Design representation

4.3.1 Models

Library elements have corresponding behavioral models for use in simulation. These models may or may not include power and ground pins for their supplies. Standard cell models are usually written as Verilog modules and use constructs such as Verilog built-in primitives or user-defined primitives (UDPs) to express the relatively simple behavior of a standard cell. They ~~may~~ [can](#) also be written as VHDL design entities (entity/architecture pairs) using package VITAL, which provides Verilog-like primitive modeling capabilities. Hard macro models ~~may~~ [can](#) be written in either language, using more complex behavioral constructs such as Verilog initial blocks and always blocks, or VHDL processes and concurrent statements.

4.3.2 Netlist

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

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

4.3.3 Behavioral models

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

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

4.3.4 HDL scopes

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

4.3.5 Design hierarchy

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

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

4.3.6 Logic hierarchy

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

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

UPF also uses the logic hierarchy as a framework for locating the power-management objects used to represent power-management concepts, e.g., power domains and power state tables (PSTs). Each such object is effectively declared in a specific scope of the logic hierarchy, and the name of the scope can be used as the prefix of the name of the object. [Furthermore, certain UPF-defined objects act as scopes themselves, in which other subordinate named objects can be defined. In such cases, the name of the UPF-defined parent object can be used as the prefix of the name of any of its subordinate objects.](#)

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

4.3.7 Hierarchy navigation

In UPF, commands are executed in the context of a scope within the logic hierarchy. The **set_scope** command (see 6.51) is used to navigate within the hierarchy and to set the current scope within which commands are executed.

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

The design top instance and design top module are typically paired: the design top instance (represented by a hierarchical name relative to the root scope) is an instance in the hierarchy representing a design for which power intent has been defined, and the design top module is the module for which the UPF file expressing this power intent has been written. The association between the UPF file and the design top module is specified in the UPF file using **set_design_top** (see 6.41); this UPF file is then typically applied to each instance of that module in a larger system.

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

The **set_scope** command (see 6.51) changes the current scope locally within the subtree depending on the current design top instance/module. Since the design top instance is typically an instance of the design top module, they both have the same hierarchical substructure; therefore, **set_scope** can be written relative to the module, but still work correctly when applied to an instance. The **set_scope** command is only allowed to change scope within this subtree. It cannot change the scope to a scope above the ~~current~~ design top instance or to a scope that is, or is below, a leaf-level instance.

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

4.3.8 Ports and nets

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

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

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

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

4.3.9 Connecting nets to ports

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

NOTE—Nets are propagated as necessary through the descendant subtree and may be renamed to avoid name collision; therefore, the same simple name in different scopes may refer to nets that are independent and unconnected.¹⁵

¹⁵ Notes in text, tables, and figures [of a standard](#) are given for information only and do not contain requirements needed to implement this standard.