BS IEC 61523-4:2015

BSI Standards Publication

Design and Verification of Low-Power Integrated Circuits

bsi.

...making excellence a habit.™

Document Sharing Hub

National foreword

This British Standard is the UK implementation of IEC 61523-4:2015.

The UK participation in its preparation was entrusted to Technical Committee EPL/501, Electronic Assembly Technology.

A list of organizations represented on this committee can be obtained on request to its secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

© The British Standards Institution 2015.

Published by BSI Standards Limited 2015

ISBN 978 0 580 88138 1

ICS 25.040; 35.060

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 30 April 2015.

Amendments/corrigenda issued since publication

Date	Text affected

Documents
Document Sharing Hub

Full document details: BS IEC 61523-4:2015

IEC 61523-4

Edition 1.0 2015-03

INTERNATIONAL STANDARD

IEEE Std 1801™-2013

Design and Verification of Low-Power Integrated Circuits

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

ICS 25.040; 35.060

ISBN 978-2-8322-2266-9

Warning! Make sure that you obtained this publication from an authorized distributor.

® Registered trademark of the International Electrotechnical Commission

Documents
Document Sharing Hub

Full document details: BS IEC 61523-4:2015

Contents

1. Overview ... 1

1.1 Scope ... 1
1.2 Purpose ... 1
1.3 Key characteristics of the Unified Power Format ... 1
1.4 Use of color in this standard ... 3
1.5 Contents of this standard ... 3

2. Normative references ... 4

3. Definitions, acronyms, and abbreviations ... 4

3.1 Definitions ... 4
3.2 Acronyms and abbreviations ... 9

4. UPF concepts ... 11

4.1 Design structure ... 11
4.2 Design representation ... 11
4.3 Power architecture ... 14
4.4 Power distribution ... 17
4.5 Power management ... 23
4.6 Power states ... 26
4.7 Simstates ... 29
4.8 Successive refinement ... 30
4.9 Tool flow ... 31
4.10 File structure ... 32

5. Language basics ... 33

5.1 UPF is Tcl ... 33
5.2 Conventions used ... 33
5.3 Lexical elements ... 34
5.4 Boolean expressions ... 37
5.5 Object declaration ... 39
5.6 Attributes of objects ... 40
5.7 Power state name spaces ... 43
5.8 Precedence ... 44
5.9 Generic UPF command semantics ... 45
5.10 effective_element_list semantics ... 45
5.11 Command refinement ... 48
5.12 Error handling ... 49
5.13 Units ... 50

6. Power intent commands ... 51

6.1 Categories ... 51
6.2 add_domain_elements [deprecated] ... 51
6.3 add_port_state [legacy] ... 52
6.4 add_power_state ... 52
6.5 add_pst_state [legacy] ... 57
6.6 apply_power_model ... 58
6.7 associate_supply_set ... 59

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: BS IEC 61523-4:2015

6.8 begin_power_model ... 60
6.9 bind_checker ... 61
6.10 connect_logic_net ... 63
6.11 connect_supply_net ... 64
6.12 connect_supply_set ... 65
6.13 create_composite_domain ... 67
6.14 create_hdl2upf_vct ... 68
6.15 create_logic_net ... 69
6.16 create_logic_port ... 70
6.17 create_power_domain ... 71
6.18 create_power_switch ... 74
6.19 create_pst [legacy] ... 80
6.20 create_supply_net ... 80
6.21 create_supply_port ... 83
6.22 create_supply_set ... 84
6.23 create_upf2hdl_vct ... 85
6.24 describe_state_transition ... 86
6.25 end_power_model ... 87
6.26 find_objects ... 88
6.27 load_simstate_behavior ... 90
6.28 load_upf ... 91
6.29 load_upf_protected ... 92
6.30 map_isolation_cell [deprecated] ... 93
6.31 map_level_shifter_cell [deprecated] ... 93
6.32 map_power_switch ... 93
6.33 map_retention_cell ... 94
6.34 merge_power_domains [deprecated] ... 97
6.35 name_format ... 98
6.36 save_upf ... 99
6.37 set_design_attributes ... 100
6.38 set_design_top ... 101
6.39 set_domain_supply_net [legacy] ... 101
6.40 set_equivalent ... 102
6.41 set_isolation ... 104
6.42 set_isolation_control [deprecated] ... 110
6.43 set_level_shifter ... 111
6.44 set_partial_on_translation ... 116
6.45 set_pin_related_supply [deprecated] ... 116
6.46 set_port_attributes ... 117
6.47 set_power_switch [deprecated] ... 121
6.48 set_repeater ... 121
6.49 set_retention ... 124
6.50 set_retention_control [deprecated] ... 128
6.51 set_retention_elements ... 128
6.52 set_scope ... 129
6.53 set_simstate_behavior ... 130
6.54 upf_version ... 131
6.55 use_interface_cell ... 132

7. Power management cell commands ... 135

7.1 Introduction ... 135
7.2 define_always_on_cell ... 136
7.3 define_diode_clamp ... 137

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

7.4 define_isolation_cell ... 138
7.5 define_level_shifter_cell ... 141
7.6 define_power_switch_cell ... 145
7.7 define_retention_cell ... 147

8. UPF processing ... 150

8.1 Overview ... 150
8.2 Data requirements ... 150
8.3 Processing phases ... 150
8.4 Error checking ... 153

9. Simulation semantics ... 154

9.1 Supply network creation ... 154
9.2 Supply network simulation ... 155
9.3 Power state simulation ... 157
9.4 Simstate simulation ... 159
9.5 Transitioning from one simstate state to another ... 161
9.6 Simulation of retention ... 162
9.7 Simulation of isolation ... 168
9.8 Simulation of level-shifting ... 168
9.9 Simulation of repeater ... 168

Annex A (informative) Bibliography ... 169

Annex B (normative) HDL package UPF ... 170

Annex C (normative) Queries ... 182

Annex D (informative) Replacing deprecated and legacy commands and options ... 219

Annex E (informative) Low-power design methodology ... 227

Annex F (normative) Value conversion tables ... 252

Annex G (normative) Supporting hard IP ... 255

Annex H (normative) UPF power-management commands semantics and Liberty mappings ... 258

Annex I (informative) Power-management cell modeling examples ... 273

Annex J (informative) Switching Activity Interchange Format ... 303

Annex K(informative) IEEE List of Participants ... 333

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: BS IEC 61523-4:2015

DESIGN AND VERIFICATION OF LOW-POWER INTEGRATED CIRCUITS

FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation.

   IEEE Standards documents are developed within IEEE Societies and Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. IEEE develops its standards through a consensus development process, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of IEEE and serve without compensation. While IEEE administers the process and establishes rules to promote fairness in the consensus development process, IEEE does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards. Use of IEEE Standards documents is wholly voluntary. IEEE documents are made available for use subject to important notices and legal disclaimers (see http://standards.ieee.org/IPR/disclaimers.html for more information).

   IEC collaborates closely with IEEE in accordance with conditions determined by agreement between the two organizations.

2) The formal decisions of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees. The formal decisions of IEEE on technical matters, once consensus within IEEE Societies and Standards Coordinating Committees has been reached, is determined by a balanced ballot of materially interested parties who indicate interest in reviewing the proposed standard. Final approval of the IEEE standards document is given by the IEEE Standards Association (IEEE-SA) Standards Board.

3) IEC/IEEE Publications have the form of recommendations for international use and are accepted by IEC National Committees/IEEE Societies in that sense. While all reasonable efforts are made to ensure that the technical content of IEC/IEEE Publications is accurate, IEC or IEEE cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.

4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications (including IEC/IEEE Publications) transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC/IEEE Publication and the corresponding national or regional publication shall be clearly indicated in the latter.

5) IEC and IEEE do not provide any attestation of conformity. Independent certification bodies provide conformity assessment services and, in some areas, access to IEC marks of conformity. IEC and IEEE are not responsible for any services carried out by independent certification bodies.

6) All users should ensure that they have the latest edition of this publication.

7) No liability shall attach to IEC or IEEE or their directors, employees, servants or agents including individual experts and members of technical committees and IEC National Committees, or volunteers of IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board, for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC/IEEE Publication or any other IEC or IEEE Publications.

8) Attention is drawn to the normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.

9) Attention is drawn to the possibility that implementation of this IEC/IEEE Publication may require use of material 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. IEC or IEEE shall not be held responsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity or scope of Patent 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.

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.

Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

International Standard IEC 61523-4/IEEE Std 1801-2013 has been processed through IEC technical committee 91: Electronics assembly technology, under the IEC/IEEE Dual Logo Agreement.

The text of this standard is based on the following documents:

IEEE Std	FDIS	Report on voting
1801 (2013)	91/1209/FDIS	91/1228/RVD

Full information on the voting for the approval of this standard can be found in the report on voting indicated in the above table.

The IEC Technical Committee and IEEE Technical Committee have decided that the contents of this publication will remain unchanged until the stability date indicated on the IEC web site under "http://webstore.iec.ch" in the data related to the specific publication. At this date, the publication will be

- reconfirmed,
- withdrawn,
- replaced by a revised edition, or
- amended.

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: BS IEC 61523-4:2015

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

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

Sponsor

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

Approved 6 March 2013

IEEE-SA Standards Board

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: BS IEC 61523-4:2015

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

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

Cadence Design Systems, Inc.
Library Cell Modeling Guide Using CPF
Hierarchical Power Intent Modeling Guide Using CPF

Silicon Integration Initiative, Inc.
Si2 Common Power Format Specification, Version 2.0

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

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

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: BS IEC 61523-4:2015

IEEE Introduction

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

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

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

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

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

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

[a]Information on references can be found in Clause 2.

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

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

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

Notice to users

Laws and regulations

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

Copyrights

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

Updating of IEEE documents

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

Errata

Errata, if any, for this and all other standards can be accessed 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.

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

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.

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: BS IEC 61523-4:2015

Design and Verification of Low-Power Integrated Circuits

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

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

1. Overview

1.1 Scope

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

1.2 Purpose

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

1.3 Key characteristics of the Unified Power Format

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

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

consistent format to specify power design information that may not be easily specifiable in an HDL or when it is undesirable to directly specify the power semantics in an HDL, as doing so would tie the logic specification directly to a constrained power implementation. UPF specifies a set of HDL attributes and HDL packages to facilitate the expression of power intent in HDL when appropriate (see Table 4 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.

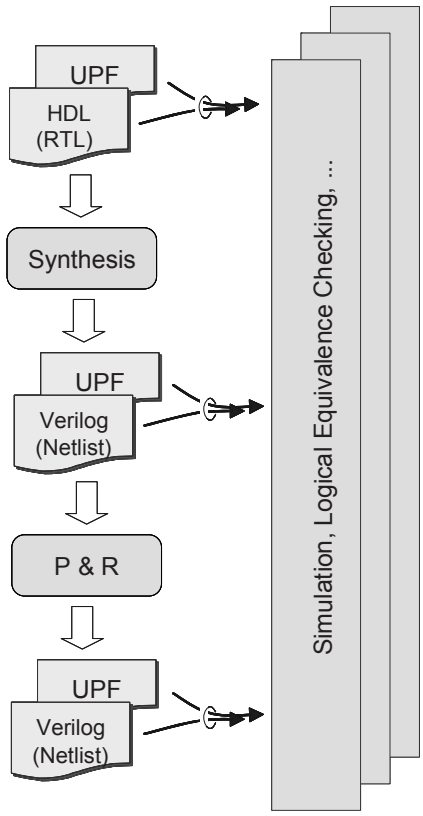

Figure 1—UPF tool flow

As indicated in Figure 1, UPF files are part of the design source 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 4.8) where power intent information will grow along the design flow to provide needed information for each design stage.

— Simulation tools can read the HDL/UPF design input files and perform RTL power-aware simulation. At this stage, the UPF may only contain abstract models such as power domains and supply sets without the need to create the power and ground network and implementation details.

— Synthesis tools can read the HDL/UPF design input files and produce a netlist. The tool or user may produce a new UPF fileset that, combined with the netlist, represents a further refined version of same design.

— In those cases where design object names change, a UPF file with the new names is needed. A UPF-aware logical equivalence checker can read the full design and UPF filesets and perform the checks to ensure power-aware equivalence.

— Place and route tools read both the netlist and the UPF files and produce a physical netlist, potentially including an output UPF file.

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

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

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

1.4 Use of color in this standard

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

— Cross references that are hyperlinked to other portions of this standard are shown in 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 provided incrementally (*layered*) are shown in **boldface-green text**. See also 5.11.

— Syntactic keywords and tokens that have been explicitly identified as legacy or deprecated constructs (see 6.1) may be shown in **brown text**.

1.5 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.

— Annexes. Following Clause 9 are a series of annexes.

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

2. Normative references

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

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

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

3. Definitions, acronyms, and abbreviations

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

3.1 Definitions

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

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

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

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

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

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

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

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

[1]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/).
[2]IEEE publications are available from The Institute of Electrical and Electronics Engineers (http://standards.ieee.org/).
[3]The IEEE standards or products referred to in this clause are trademarks of The Institute of Electrical and Electronics Engineers, Inc.
[4]*IEEE Standards Dictionary Online* subscription is available at:
http://www.ieee.org/portal/innovate/products/standard/standards_dictionary.html.
[5]The numbers in brackets correspond to those of the bibliography in Annex A.
[6]Information on references can be found in Clause 2.

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

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

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

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

connected: Attached together via a direct connection.

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

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

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

NOTE—See 6.52.[7]

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

descendant: Any **instance** between the **current scope** in the **logic hierarchy** and its **leaf-level instance**s. 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 **descendant**s.

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

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

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

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

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

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

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

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

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

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

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

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

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

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

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

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

golden UPF: *See:* **configuration UPF**.

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

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

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

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

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

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

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

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

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

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 **descendant**s, or an **instance** that has the attribute **UPF_is_leaf_cell** associated with it.

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.

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

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

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

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

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

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

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

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

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

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

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

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

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

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

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

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

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

NOTE—See Clause 4.

power state table (PST): A table that captures the legal combinations of **power state**s for a set of **supply port**s and/or **supply net**s.

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

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

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

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

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

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

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

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

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

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

NOTE—See 6.7 and 6.11.

scope: An **instance** in the **logic hierarchy**.

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

silicon UPF: *See:* **implementation UPF**.

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

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

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

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

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

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

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

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

supply net: An HDL representation of a power rail.

supply port: A connection point for **supply net**s.

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

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

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

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

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

3.2 Acronyms and abbreviations

CMOS complementary metal oxide semiconductor

DFT Design for Test

EDA electronic design automation

HDL hardware description language

IP intellectual property

MSFF master-slave flip-flop

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

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

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

4. UPF concepts

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

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

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

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

4.1 Design structure

4.1.1 Transistors

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

4.1.2 Standard cells

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

4.1.3 Hard macros

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

4.2 Design representation

4.2.1 Models

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

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

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

4.2.2 Netlist

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

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

4.2.3 Behavioral models

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

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

4.2.4 HDL scopes

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

4.2.5 Design hierarchy

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

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

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

4.2.6 Logic hierarchy

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

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

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

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

4.2.7 Hierarchy navigation

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

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

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

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

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

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

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

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

4.2.8 Ports and nets

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

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

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

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

4.2.9 Connecting nets to ports

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

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

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

4.3 Power architecture

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

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

4.3.1 Power domains

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

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

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

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

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

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

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

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

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

4.3.2 Drivers, receivers, sources, and sinks

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

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

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

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

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

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

4.3.3 Isolation and level-shifting

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

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

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

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

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

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

4.3.4 State retention

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

For a retention register, the following terms apply:

— *Register value* is the data held in the storage element of the register. In functional mode, this value gets updated on the rising/falling edge of clock or gets set or cleared by set/reset signals, respectively.

— *Retained value* is the data in the retention element of retention register. The retention element is powered by the retention supply.

— *Output value* is the value on the output of the register.

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

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

b) *Master/slave-alive retention*: In a master/slave-alive retention register, the retained value is held in the master or slave latch. In this case, the retention element is in the functional data-path of the register.

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

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

4.4 Power distribution

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

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

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

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

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

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

4.4.1 Supply network elements

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

4.4.1.1 Supply ports and nets

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

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

4.4.1.2 Supply switches

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

4.4.1.3 Supply sets

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

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

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

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

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

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

4.4.2 Supply network construction

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

4.4.2.1 Supply sources and loads

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

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

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

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

4.4.2.2 Supply port/net connections

Connections are made from nets to ports

a) from a net to (the LowConn of) a port declared in the same scope; or

b) from a net to (the HighConn of) a port declared in a lower scope; or

c) from a net to a pin of a leaf cell.

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

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

4.4.2.3 Supply net resolution

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

— An unresolved supply net shall have only one supply source connection.

— A resolved supply net may have multiple supply source connections. The resolution type may restrict how many supply sources can be on at the same time.

A supply net may have any number of load connections.

4.4.2.4 Supply net / supply set connections

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

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

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

4.4.2.5 Supply set function connections

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

4.4.2.5.1 Explicit and automatic connections

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

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

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

a) **power** is connected by default to ports having the *pg_type* `primary_power`.

b) **ground** is connected by default to ports having the *pg_type* `primary_ground`.

c) **pwell** is connected by default to ports having the *pg_type* `pwell`.

d) **nwell** is connected by default to ports having the *pg_type* `nwell`.

e) **deeppwell** is connected by default to ports having the *pg_type* `deeppwell`.

f) **deepnwell** is connected by default to ports having the *pg_type* `deepnwell`.

4.4.2.5.2 Implicit connections

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

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

a) Primary supply set

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

b) Retention supply set

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

c) Isolation supply set

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

d) Level-shifter supply sets

The functions of a supply set for a level-shifting strategy are implicitly connected as appropriate to the input, output, or internal supply pins of any level-shifter implied by the application of the strategy.

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

After UPF-specified power intent has been completely applied, it shall be an error if any instance in the design does not have a supply set function or supply net connected to each of its supply ports, including any implicit power and ground ports.

4.4.2.6 Supply set required functions

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

— `power` and `ground` are typically required in all cases.

— `nwell`, `pwell`, `deepnwell`, and `deeppwell` are only required occasionally.

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

a) Any function used to define a power state of the supply set,

b) Any function used for automatic connection of the supply set based on *pg_type,* and

c) Any required function of a supply set handle with which the supply set is associated.

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

4.4.3 Supply equivalence

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

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

Electrical equivalence can affect

a) the number of sources of a supply network, and therefore,

b) whether resolution is required for that supply network.

Electrical equivalence implies functional equivalence, but not vice versa.

Functional equivalence can affect any of the following:

c) Insertion of isolation cells, level-shifter cells, and repeater cells

d) Determination of power-domain lower boundaries

e) Legality of power-domain composition

f) Validity of driver and receiver supply attributes

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

4.4.3.1 Supply port/net equivalence

Electrical equivalence is determined by connection, as follows:

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

a) A port P is electrically equivalent to itself.

b) A net N is electrically equivalent to itself.

c) If a net N and a port P are connected, then N and P are electrically equivalent.

d) If A and B are electrically equivalent, and B and C are electrically equivalent, then A and C are electrically equivalent.

e) If A and B are connected via a supply set function (see 4.4.2.4), then A and B are electrically equivalent.

Electrical equivalence can also be declared, as follows:

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

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

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

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

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

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

4.4.3.2 Supply set equivalence

A supply set handle is also a supply set.

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

Corresponding functions of two supply sets are electrically equivalent if

— their associated supply nets are electrically equivalent, or

— the two supply sets are directly associated with one another.

Corresponding functions of two supply sets are functionally equivalent if

— they are electrically equivalent, or

— they have been declared as functionally equivalent.

Two supply sets are (functionally) equivalent if

— they both have the same required functions, and the nets associated with corresponding functions are equivalent; or

— they are associated with each other directly or indirectly via one or more **associate_supply_set** commands (see 6.7); or

— they are each associated directly or indirectly via **associate_supply_set** (see 6.7) with two other supply sets, which are equivalent.

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

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

As a consequence of this,

a) two anonymous supply sets built from equivalent PG functions are equivalent;

b) two supply sets that are functionally equivalent can be used interchangeably;

c) a supply set and any supply set handle it is associated with are always equivalent.

4.5 Power management

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

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

4.5.1 Related supplies

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

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

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

4.5.2 Driver and receiver supplies

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

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

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

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

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

4.5.3 Logic sources and sinks

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

— The LowConn of an input or inout logic port whose HighConn is connected to an external driver is a source.

— The HighConn of an output or inout logic port whose LowConn is connected to an internal driver is a source.

— The LowConn of an output or inout logic port whose Highconn is connected to an external receiver is a sink.

— The Highconn of an input or inout logic port whose LowConn is connected to an internal receiver is a sink.

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

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

4.5.4 Power-management requirements

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

— If a power domain is powered down in certain situations, its state registers may need to have their values saved before power-down and restored after subsequent power-up, either to maintain persistent data or to enable faster power-up.

— If the distance between driver and receiver is long (the capacitive load is high), buffers (repeaters) may be required to strengthen the signal along the way, or to ensure that it stabilizes within the required time.

— If a receiver is powered on, but its driver is not, an isolation cell is required between driver and receiver to drive the receiver with a known value despite the fact that the ultimate driver is powered off.

— If the driver and receiver supplies (or isolation and receiver supplies, or driver and isolation supplies, etc.) are operating at different voltage levels, a level-shifter is required between them to translate between voltage levels.

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

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

4.5.5 Power-management strategies

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

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

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

4.5.6 Power-management implementation

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

a) Strategies are implemented in the following order: retention strategies, followed by repeater strategies, followed by isolation strategies, followed by level-shifter strategies.

b) A retention strategy may affect the driving supply of the retention cell output. If so, the new driving supply of the retention cell is visible to, and affects the result of, a source filter of any subsequently applied strategy.

c) A repeater strategy causes insertion of a buffer, which has a receiver and a driver; this insertion therefore affects both the receiving supply of ports driving the repeater input and the driving supply of ports receiving the repeater output. The new driving supply and receiver supply are visible to, and affect the result of, source and sink filters, respectively, of any subsequently applied strategy.

d) An isolation strategy may cause insertion of an isolation cell, which has a receiver and a driver; therefore if such insertion occurs, it affects both the receiving supply of ports driving the isolation cell input and the driving supply of ports receiving the isolation cell output. However, the new driving supply and receiver supply are not visible to, and do not affect the result of, source and sink filters, respectively, of any subsequently applied isolation or level-shifting strategies.

e) A level-shifting strategy may cause insertion of a level-shifting cell, which has a receiver and a driver; therefore if such insertion occurs, it affects both the receiving supply of ports driving the level-shifting cell input and the driving supply of ports receiving the level-shifting cell output. However, the new driving supply and receiver supply are not visible to, and do not affect the result of, source and sink filters, respectively, of any subsequently applied level-shifting strategy.

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

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

4.5.7 Power control logic

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

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

4.6 Power states

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

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

4.6.1 Power state of a supply port or supply net

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

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

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

4.6.2 Power state of a supply set

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

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

— Whether there is current available to power an instance, and

— The voltage level of the supply.

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

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

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

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

4.6.3 Predefined supply set power states

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

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

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

4.6.4 Power states of power domains

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

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

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

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

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

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

4) The isolation enable(s) are active.

5) The retention control signal(s) are active.

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

4.6.5 Power states of systems and subsystems

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

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

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

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

4.6.6 Incremental refinement of power states

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

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

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

4.7 Simstates

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

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

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

a) **CORRUPT**—The supply set is either off (one or more supply nets in the set are switched off, terminating the flow of current) or at such a low-voltage level that it cannot support switching and the retention of the state of logic nets cannot be guaranteed to be maintained even in the absence of activity in the instances powered by the supply.

b) **CORRUPT_ON_ACTIVITY**—The power characteristics of the supply set are sufficient for logic nets to retain their state as long as there is no activity within the elements connected to the supply, but they are insufficient to support activity.

c) **CORRUPT_ON_CHANGE**—The power characteristics of the supply set are sufficient for logic nets to retain their state as long as there is no change in the outputs of the elements connected to that supply.

d) **CORRUPT_STATE_ON_ACTIVITY**—The power characteristics of the supply set are sufficient to support normal operation of combinational logic, but they are insufficient to support activity inside state elements, whether that activity would result in any state change or not.

e) **CORRUPT_STATE_ON_CHANGE**—The power characteristics of the supply set are sufficient to support normal operation of combinational logic, and they are sufficient to support activity inside state elements, but they are insufficient to support a change of state for state elements.

f) **NORMAL**—The power characteristics of the supply set are sufficient to support full and complete operational (switching) capabilities with characterized timing.

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

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

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

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

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

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

4.8 Successive refinement

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

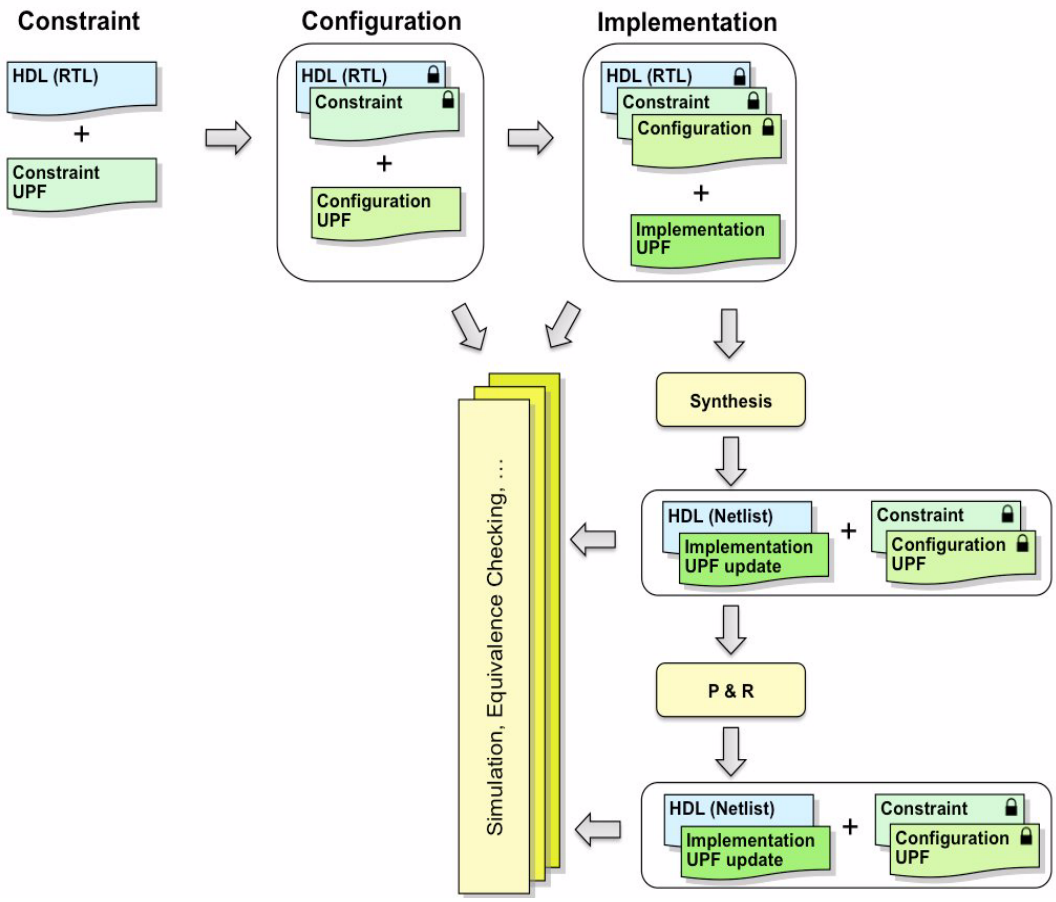

Figure 2—Successive refinement of power intent

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

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

a) The atomic power domains in the design.

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

b) The state variables that need to have their values retained if a given power domain is powered down.

This does not involve specifying how such retention would be controlled. [Use **set_retention_elements** (see 6.51).]

c) The clamp values of signals that would need to be isolated if a given power domain is powered down.

This does not involve specifying how isolation is to be controlled. [Use **set_port_attributes –clamp_value** (see 6.46).]

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

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

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

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

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

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

4.9 Tool flow

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

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

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

— By aspect: isolation, level-shifting, retention, repeaters, control logic, power distribution

— By command: isolation strategy A, isolation strategy B, etc.

— By element to which a command applies: isolation for port `p1`, for port `p2`, etc.

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

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

a) What part of the UPF power intent specification is supposed to be implemented already, and

b) What part of the UPF power intent specification is to be included in the processing done by this tool.

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

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

4.10 File structure

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

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

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

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

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

5. Language basics

5.1 UPF is Tcl

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

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

— UPF power intent commands are executed in the order of occurrence, just as Tcl commands are executed and return values can be used by subsequent commands.

— The only UPF commands that support regular expressions are **find_objects** (see 6.26) and **query_upf** (see C.1).

— All of the commands and techniques of Tcl may be used, including procs and libraries of procs. However, the procs and libraries of procs should, in the end, only rely on UPF commands for design information.

— **find_objects** (see 6.26) shall be the only source used to programmatically access the HDL when defining the power intent. The processing of information returned by **find_objects** using standard Tcl commands [B5], such as `regexp`, is allowed.

— UPF is intended to be used across many tools, so it is erroneous to use proprietary tool specific commands when constructing power intent.

— Once the Tcl processing has completed, the end result can be expressed as a series of UPF commands.

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

5.2 Conventions used

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

Table 1—Document conventions

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

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

Table 1—Document conventions *(continued)*

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

5.3 Lexical elements

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

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

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015

5.3.1 Identifiers

Identifiers adhere to the following rules:

a) The first character of a identifier shall be alphabetic.

b) All other characters of a identifier shall be alphanumeric or the underscore character (_).

c) Identifiers in UPF are case-sensitive.

5.3.2 Keywords and reserved words

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

a) Domain record field space
   1) **primary**
   2) **default_retention**
   3) **default_isolation**

b) Switch record field space
   1) **supply**

c) Level-shifter strategy record field name space
   1) **input_supply_set**
   2) **output_supply_set**
   3) **internal_supply_set**

d) Isolation strategy record field name space
   1) **isolation_supply_set**
   2) **isolation_signal**

e) Retention strategy record field name space (see 6.33)
   1) **retention_ref**
   2) **retention_supply_set**
   3) **primary_ref**
   4) **primary_supply_set**
   5) **save_signal**
   6) **restore_signal**
   7) **UPF_GENERIC_CLOCK**
   8) **UPF_GENERIC_DATA**
   9) **UPF_GENERIC_ASYNC_LOAD**
   10) **UPF_GENERIC_OUTPUT**

5.3.3 Names

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

5.3.3.1 Simple names

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

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

Published by IEC under license from IEEE. © 2013 IEEE. All rights reserved.
Documents
Document Sharing Hub
Full document details: BS IEC 61523-4:2015