

IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows

IEEE Computer Society

Sponsored by the
Design Automation Standards Committee

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

IEEE Std 1685™-2014
(Revision of
IEEE Std 1685-2009)

IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows

Sponsor

**Design Automation Standards Committee
of the
IEEE Computer Society**

Approved 12 June 2014

IEEE-SA Standards Board

Contributed updates to IEEE Std 1685-2009 from the Accellera Systems Initiative IP-XACT Working Group are acknowledged.

Abstract: Conformance checks for eXtensible Markup Language (XML) data designed to describe electronic systems are formulated by this standard. The meta-data forms that are standardized include components, systems, bus interfaces and connections, abstractions of those buses, and details of the components including address maps, register and field descriptions, and file set descriptions for use in automating design, verification, documentation, and use flows for electronic systems. A set of XML schemas of the form described by the World Wide Web Consortium (W3C®) and a set of semantic consistency rules (SCRs) are included. A generator interface that is portable across tool environments is provided. The specified combination of methodology-independent meta-data and the tool-independent mechanism for accessing that data provides for portability of design data, design methodologies, and environment implementations.

Keywords: abstraction definitions, address space specification, bus definitions, design environment, EDA, electronic design automation, electronic system level, ESL, IEEE 1685™, implementation constraints, IP-XACT, register transfer level, RTL, SCRs, semantic consistency rules, TGI, tight generator interface, tool and data interoperability, use models, XML design meta-data, XML schema

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

Copyright © 2014 by The Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 12 September 2014. Printed in the United States of America.

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

AMBA is a registered trademark of ARM Limited.
SystemC is a registered trademark of Open SystemC Initiative, Inc.
Verilog is a registered trademark of Cadence Design Systems, Inc.
W3C is a registered trademark of the World Wide Web Consortium.
XMLSpy is a registered trademark of Altova GmbH.

PDF: ISBN 978-0-7381-9226-0 **STD98727**
Print: ISBN 978-0-7381-9227-7 **STDPD98727**

IEEE prohibits discrimination, harassment, and bullying.

For more information, visit <http://www.ieee.org/web/aboutus/whatis/policies/9-26.html>.

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

Important Notices and Disclaimers Concerning IEEE Standards Documents

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

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

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

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

Use of an IEEE standard is wholly voluntary. The existence of an IEEE standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard.

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

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

Translations

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

Official statements

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

Comments on standards

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

Comments on standards should be submitted to the following address:

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

Laws and regulations

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

Copyrights

IEEE draft and approved standards are copyrighted by IEEE under U.S. and international copyright laws. They are made available by IEEE and are adopted for a wide variety of both public and private uses. These include both use, by reference, in laws and regulations, and use in private self-regulation, standardization, and the promotion of engineering practices and methods. By making these documents available for use and adoption by public authorities and private users, IEEE does not waive any rights in copyright to the documents.

Photocopies

Subject to payment of the appropriate fee, IEEE will grant users a limited, non-exclusive license to photocopy portions of any individual standard for company or organizational internal use or individual, non-commercial use only. To arrange for payment of licensing fees, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Updating of IEEE Standards documents

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

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

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

Errata

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

Patents

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

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

Participants

The IP-XACT Standardization Working Group is entity based. At the time this standard was completed, the IP-XACT Standardization Working Group had the following membership:

Mark Noll, *Chair*
Erwin de Kock, *Vice Chair*
David Courtright, *Secretary*
Joe Daniels, *Technical Editor*

David Courtright
Joe Daniels
Edwin Dankert
Jean-Michel Fernandez

Jeffery Griffiths
Stephane Guntz
Prashant Karandikar

Erwin de Kock
Mark Noll
Kamlesh Pathak
Richard Weber

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

Accellera Organization, Inc.
Advanced Micro Devices (AMD)
Cadence Design Systems, Inc.
Duolog Ltd.

MAGILLEM LLC
Mentor Graphics
NXP Semiconductors

Semifore, Inc.
STMicroelectronics
Synopsys, Inc.
Texas Instruments Incorporated

When the IEEE-SA Standards Board approved this standard on 12 June 2014, it had the following membership:

John Kulick, *Chair*
Jon Walter Rosdahl, *Vice-chair*
Richard H. Hulett, *Past Chair*
Konstantinos Karachalios, *Secretary*

Peter Balma
Farooq Bari
Ted Burse
Clint Chaplain
Stephen Dukes
Jean-Phillippe Faure
Gary Hoffman

Michael Janezic
Jeffrey Katz
Joseph L. Koepfinger*
David Law
Hung Ling
Oleg Logvinov
Ted Olsen
Glenn Parsons

Ron Peterson
Adrian Stephens
Peter Sutherland
Yatin Trivedi
Phil Winston
Don Wright
Yu Yuan

* Member Emeritus

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

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

Don Messina
IEEE-SA Content Publishing

Joan Woolery
IEEE-SA Standards Technical Community

Introduction

This introduction is not part of IEEE Std 1685-2014, IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows.

The purpose of this standard is to provide the electronic design automation (EDA), semiconductor, electronic design intellectual property (IP) provider, and system design communities with a well-defined and unified specification for the meta-data that represents the components and designs within an electronic system. The goals of this specification are to enable delivery of compatible IP descriptions from multiple IP vendors; to improve the importing and exporting of complex IP bundles to, from, and between EDA tools for system on chip (SoC) design environments (DEs); to improve the expression of configurable IP by using IP meta-data; and to improve the provision of EDA vendor-neutral IP creation and configuration scripts (*generators*). The data and data access specification is designed to coexist and enhance the hardware description languages (HDLs) presently used by designers while providing capabilities lacking in those languages.

The SPIRIT Consortium, which originally developed the IP-XACT standard before merging with Accellera, was a consortium of electronic system, IP provider, semiconductor, and EDA companies. IP-XACT enables a productivity boost in design, transfer, validation, documentation, and use of electronic IP and covers components, designs, interfaces, and details thereof. The data specified by IP-XACT is extensible in locations specified in the schema.

IP-XACT enables the use of a unified structure for the meta specification of a design, components, interfaces, documentation, and interconnection of components. This structure can be used as the basis of both manual and automatic methodologies. IP-XACT specifies the tight generator interface (TGI) for access to the data in a vendor-independent manner.

This standardization project provides electronic design engineers with a well-defined standard that meets their requirements in structured design and validation and enables a step function increase in their productivity. This standardization project will also provide the EDA industry with a standard to which they can adhere and that they can support in order to deliver their solutions in this area.

Accellera has prepared a set of bus and abstraction definitions for several common buses. It is expected, over time, that standards groups and manufacturers who define buses will include IP-XACT eXtensible Markup Language (XML) bus and abstraction definitions in their set of deliverables. Until that time, and to cover existing useful buses, a set of bus and abstraction definitions for common buses has been created.

A set of reference bus and abstraction definitions allows many vendors who define IP using these buses to easily interconnect IP together. Accellera posts these definitions for use by its members, with no warranty of suitability, but in the hope that they will be useful. Accellera will, from time to time, update these files and, if a standards body wishes to take over the work of definition, will transfer that work to that body.

These reference bus and abstraction definition templates (with comments and examples) are available from the public area of the Accellera Web site.^a

^aAvailable at <http://www.accellera.org>.

Contents

1.	Overview	1
1.1	Scope	1
1.2	Purpose	2
1.3	Design environment	2
1.4	IP-XACT-enabled implementations	6
1.5	Conventions used	7
1.6	Use of color in this standard	12
1.7	Contents of this standard	12
2.	Normative references	13
3.	Definitions, acronyms, and abbreviations	15
3.1	Definitions	15
3.2	Acronyms and abbreviations	20
4.	Interoperability use model	21
4.1	Roles and responsibilities	21
4.2	IP-XACT IP exchange flows	22
5.	Interface definition descriptions	25
5.1	Definition descriptions	25
5.2	Bus definition	26
5.3	Abstraction definition	28
5.4	Ports	29
5.5	Wire ports	30
5.6	Qualifiers	31
5.7	Wire port group	32
5.8	Wire port mode (and mirrored mode) constraints	33
5.9	Transactional ports	34
5.10	Transactional port group	36
5.11	Extending bus and abstraction definitions	37
5.12	Clock and reset handling	38
6.	Component descriptions	41
6.1	Component	41
6.2	Interfaces	43
6.3	Interface interconnections	44
6.4	Complex interface interconnections	46
6.5	Bus interfaces	48
6.6	Indirect interfaces	58
6.7	Component channels	59
6.8	Address spaces	60
6.9	Memory maps	68
6.10	Remapping	79
6.11	Registers	81
6.12	Models	95

6.13	Component generators.....	122
6.14	Choices	124
6.15	File sets.....	125
6.16	White box elements.....	131
6.17	White box element reference.....	133
6.18	CPUs.....	134
6.19	Reset types.....	135
7.	Design descriptions.....	137
7.1	Design.....	137
7.2	Design component instances	138
7.3	Design interconnections	139
7.4	Active, hierarchical, monitored, and monitor interfaces	141
7.5	Design ad hoc connections	144
7.6	Port references.....	146
8.	Abstractor descriptions	149
8.1	Abstractor	149
8.2	Abstractor interfaces	151
8.3	Abstractor models	152
8.4	Abstractor views.....	152
8.5	Abstractor ports	153
8.6	Abstractor wire ports.....	155
8.7	Abstractor generators	156
9.	Generator chain descriptions	159
9.1	generatorChain	159
9.2	generatorChainSelector	161
9.3	generatorChain component selector	162
9.4	generatorChain generator	163
10.	Design configuration descriptions	165
10.1	Design configuration	165
10.2	designConfiguration	166
10.3	interconnectionConfiguration.....	167
10.4	abstractorInstance.....	168
10.5	viewConfiguration.....	169
11.	Catalog descriptions.....	171
11.1	catalog	171
11.2	ipxactFile.....	173
12.	Addressing	175
12.1	Calculating the bit address of a bit in a memory map	175
12.2	Calculating the bus address at the slave bus interface	177
12.3	Calculating the address at the indirect interface.....	177
12.4	Address modifications of a channel	178
12.5	Addressing in the master	178

12.6	Address translation in a bridge.....	178
13.	Data visibility.....	180
13.1	Mapped address bits mask.....	180
13.2	Address modifications of an interconnection.....	180
13.3	Bit steering in a channel.....	180
13.4	Visibility of bits.....	181
Annex A	(informative) Bibliography	185
Annex B	(normative) Semantic consistency rules.....	187
Annex C	(normative) Common elements and concepts	217
Annex D	(normative) Types	251
Annex E	(normative) SystemVerilog expressions.....	253
Annex F	(normative) Tight generator interface.....	267
Annex G	(informative) External bus with an internal/digital interface	459
Annex H	(informative) Bridges and channels	461
Annex I	(informative) Examples	471

IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows

IMPORTANT NOTICE: IEEE Standards documents are not intended to ensure safety, security, health, or environmental protection, or ensure against interference with or from other devices or networks. Implementers of IEEE Standards documents are responsible for determining and complying with all appropriate safety, security, environmental, health and interference protection practices and all applicable laws and regulations.

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

This clause explains the scope and purpose of this standard; gives an overview of the basic concepts, major semantic components, and conventions used in this standard; and summarizes its contents.

1.1 Scope

This standard describes an eXtensible Markup Language (XML) schema¹ for meta-data documenting *intellectual property* (IP) used in the development, implementation, and verification of electronic systems. This schema provides both a standard method to document IP that is compatible with automated integration techniques and a standard method (generators) for linking tools into a *system development* framework, enabling a more flexible, optimized development environment. Tools compliant with this standard will be able to interpret, configure, integrate, and manipulate IP blocks that comply with the IP meta-data description. The standard is independent of any specific design processes. It does not cover behavioral characteristics of the IP that are not relevant to integration.

¹Information on references can be found in [Clause 2](#).

1.2 Purpose

This standard enables the creation and exchange of IP in a highly automated design environment.

1.3 Design environment

The IP-XACT specification is a mechanism to express and exchange information about design IP and its required configuration.² While the IP-XACT description formats are the core of this standard, describing the IP-XACT specification in the context of its basic use model, the design environment (DE), more readily depicts the extent and limitations of the semantic intent of the data. The DE coordinates a set of tools and IP, or expressions of that IP (e.g., models), through the creation and maintenance of meta-data descriptions of the system on chip (SoC) so that its system design and implementation flows are efficiently enabled and reuse centric.

The use of the IP-XACT-specified formats and interfaces are shown, in **bold**, in [Figure 1](#) and described in the following subclauses.

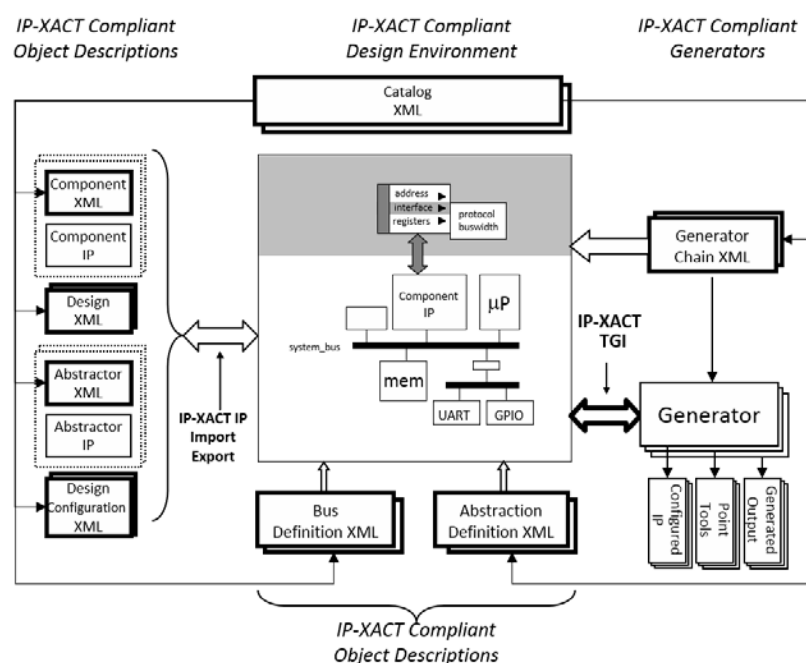


Figure 1—IP-XACT design environment

1.3.1 IP-XACT design environment

A DE enables the designer to work with IP-XACT design IP through a coordinated front-end and IP design database. These tools create and manage the top-level meta-description of system design and may provide two basic types of services: *design capture*, which is the expression of design configuration by the IP provider and design intent by the IP user, and *design build*, which is the creation of a design (or design model) to those intentions.

²IP-XACT uses the World Wide Web Consortium (W3C[®]) standard for the XML version 1.0 data (<http://www.w3.org/TR/REC-xml/>). The valid format of that XML data is described in a *schema* by using the Schema Description Language described therein. W3C is a registered trademark of the World Wide Web Consortium.

As part of design capture, a system design tool shall recognize the structure and configuration options of imported IP. In the case of *structure*, this implies both the structure of the design (e.g., how specific pin-outs refer to lines in the hardware description language (HDL) code) as well as the structure of the IP package (e.g., where design descriptions and related generators are provided in the packaged IP data-structure). In the case of *configuration*, this is the set of options for handling the imported IP (e.g., setting the base address and offset, bus width) that may be expressed as configurable parameters in the IP-XACT meta-data.

As part of design build, generators may be provided internally by a system design tool to achieve the required IP integration or configuration, or they may be provided externally (e.g., by an IP provider) and launched by the system design tool as appropriate.

The *system design tool set* defines a DE where the support for conceptual context and management of IP-XACT meta-data resides. However, the IP-XACT specifications make no requirements upon system design tool architecture or a tool's internal data structures. To be considered IP-XACT enabled, a system design tool shall support the import/export of IP expressed with valid IP-XACT meta-data for both component IP and designs, and it needs to support the tight generator interface (TGI) for interfacing with external generators (to the DE).

1.3.2 IP-XACT object descriptions

The IP-XACT schema is the core of the IP-XACT specification. There are eight top-level schema definitions. Each schema definition can be used to create object descriptions of the corresponding types:

- A *bus definition* description defines the type attributes of a bus.
- An *abstraction definition* description defines the representation attributes of a bus.
- A *component* description defines an IP or interconnect structure.
- A *design* description defines the configuration of and interconnection between components.
- An *abstractor* description defines an adaptor between interfaces of two different abstractions.
- A *generator chain* description defines the grouping and ordering of generators.
- A *design configuration* description defines additional configuration information for a generator chain or design description.
- A *catalog* description provides a mapping between IP-XACT VLNVs (see [1.3.3](#)) and the physical location of the IP-XACT file defining the IP-XACT object with the given VLNV.

1.3.3 Object interactions

An object description contains a unique identifier in the header. The identifier in IP-XACT terms is called a *VLNV* after the four elements that define its value: vendor, library, name, and version. See [C.25](#) for further details on a VLNV. This VLNV is used to create a reference from one description to another. The links between these objects are illustrated in [Figure 2](#). The arrows ($A \rightarrow B$) illustrate a reference of one object to another (e.g., reference of object B from object A).

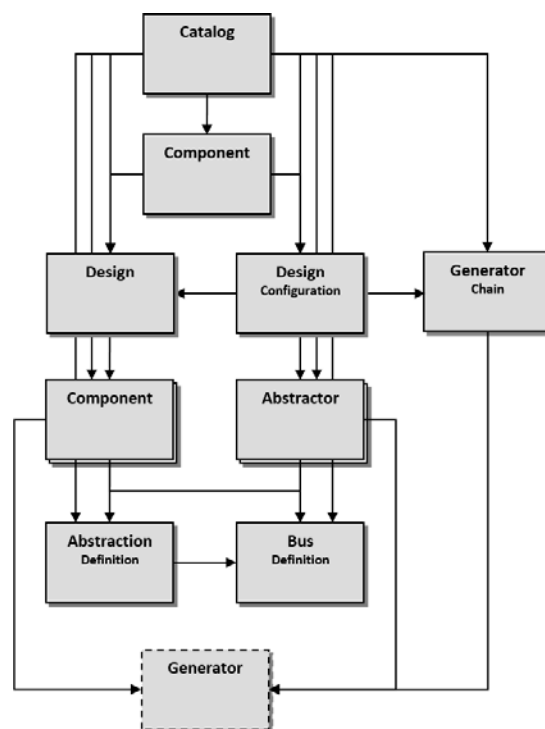


Figure 2—IP-XACT object interactions

1.3.4 IP-XACT generators

Generators are executable objects (e.g., scripts or binary programs) that may be integrated within a DE (referred to as *internal*) or provided separately as an executable (referred to as *external*). Generators may be provided as part of an IP package (e.g., for configurable IP, such as a bus-matrix generator) or as a way of wrapping point tools for interaction with a DE (e.g., an external design netlist, external design checker).

An internal generator may perform a wide variety of tasks and may access IP-XACT-compliant meta-data by any method a DE supports. IP-XACT does not describe these protocols.

An *external generator* (often referred to as a *TGI generator*) is an executable program or script invoked from within a DE to query or configure design descriptions and their related component and abstractor descriptions. External generators can use the TGI to *access* their IP-XACT meta-data descriptions (as currently loaded into the DE) and perform the various operations associated with those descriptions. In addition, external generators shall only *operate* upon IP-XACT-compliant meta-data through the defined TGI; see [1.3.6](#).

Generators can be referenced from a component, abstractor, or generator chain description. Generators can also be grouped and ordered in generator chain descriptions and in chain descriptions contained inside other chain descriptions. This sequencing of generators is *critical* for providing script-based support for SoC flow creation.

1.3.5 IP-XACT design environment interfaces

There are two interfaces expressed in [Figure 1](#): from the DE to the external IP libraries and from the DE to the generators. In the former case, the IP-XACT specifications are *neutral* regarding the design tool

interfaces to IP repositories. Being able to read and write IP with IP-XACT meta-data is required; however, the *formal interaction* between an external IP repository and a DE is not specified. In the latter case, the interface between the DE and a generator is the TGI; see [1.3.6](#).

1.3.6 Tight generator interface

The *tight generator interface* (TGI) is the method a generator uses to access a design or component description in a DE-independent and generator-language-independent manner. Therefore, a generator running on two different DEs produces the same results. The DE and the generator communicate with each other by sending messages utilizing the Simple Object Access Protocol (SOAP)³ specified in the Web Services Description Language (WSDL).⁴ SOAP provides a simple means for sending XML-format messages using the Hypertext Transfer Protocol (HTTP) or other transport protocols. IP-XACT supports using HTTP or a file protocol.

The SOAP messages passed between the generator and the DE allow the generator to get all information about the design interconnections (which contain components and abstractors), provide set information for any configurable elements in a component or abstractor, and make simple modifications of the design description. For additional details on the DE generator invocation and the SOAP messages passed between the generator and the DE, see [Annex F](#).

1.3.7 Design intellectual property

IP-XACT is structured around the concept of IP reuse. *Electronic design intellectual property*, or IP, is a term used in the electronic design automation (EDA) community to refer to a reusable collection of design specifications that represent the behavior, properties, and/or description of the design in various media. The name IP is partially derived from the common practice of considering a collection of this type to be the intellectual property of one party. Both hardware and software collections are encompassed by this term.

These collections may include the following:

- a) Design objects—This collection can include the following:
 - 1) Transaction-level modeling (TLM) descriptions: SystemC[®] and SystemVerilog⁵
 - 2) Fixed HDL descriptions: Verilog[®], VHDL⁶
 - 3) Configurable HDL descriptions (e.g., bus-fabric generators)
 - 4) Design models for register transfer level (RTL) and transactional simulation (e.g., compiled core models)
 - 5) HDL-specified verification IP (VIP) (e.g., basic stimulus generators and checkers)
- b) IP views—This collection is a list of different views (levels of description and/or languages) to describe the IP object. In IP-XACT, these views include the following:
 - 1) Design view: RTL Verilog or VHDL, flat or hierarchical components
 - 2) Simulation view: model views, targets, simulation directives, etc.
 - 3) Documentation view: standard, user guide, etc.

³Available from the W3C Web site at <http://www.w3.org/TR/soap12-part1/>.

⁴Available from the W3C Web site at <http://www.w3.org/TR/wsdl>.

⁵SystemC is a registered trademark of Open SystemC Initiative, Inc. in the United States and other countries. Verilog is a registered trademark of Cadence Design Systems, Inc. in the United States and/or other jurisdictions. This information is given for the convenience of users of this standard and does not constitute an endorsement by the IEEE of these products. Equivalent products may be used if they can be shown to lead to the same results.

⁶See Footnote 5.

IP-XACT XML meta-data descriptions provide a standardized way of collecting much of the structural information contained in the file sets. IP-XACT also can contain the information that identifies the appropriate files included in a collection to be used for different parts of the design process.

1.4 IP-XACT–enabled implementations

Complying with the rules outlined in this subclause allows providers of tools, IP, or generators to class their products as *IP-XACT enabled*. Conversely, any violation of these rules removes that naming right. This subclause first introduces the set of metrics for measuring the valid use of the specifications. It then specifies when those validity checks are performed by the various classes of products and providers: DEs, point tools, IPs, and generators.

- a) Parse validity
 - 1) Parsing correctness: Ability to read all IP-XACT descriptions.
 - 2) Parsing completeness: Cannot require information that could be expressed in an IP-XACT format to be specified in a non-IP-XACT format. Processing of all information present in an IP-XACT document is not required.
- b) Description validity
 - 1) Schema correctness: IP is described using XML files that conform to the IP-XACT schema.
 - 2) Usage completeness: Extensions to the IP-XACT schema shall be used only to express information that cannot otherwise be described in IP-XACT.
- c) Semantic validity
 - 1) Semantic correctness: Adheres to the semantic interpretations of IP-XACT data described in this standard.
 - 2) Semantic completeness: Obeys all the semantic consistency rules (SCRs) described in [Annex B](#).

These validity rules can be combined with the product class-specific rules to cover the full IP-XACT–enabled space. The following subclauses describe the rules a provider has to check to claim a product is IP-XACT enabled.

An IP-XACT–enabled DE or point tool may read descriptions based on multiple versions of the IP-XACT schema. If the DE or point tool does provide this capability, the effect shall be as if all of the descriptions had been translated to the highest schema version supported by the given tool. This is required for semantic consistency. Schema version translation can be done in a number of different ways, but the most common is to leverage the eXtensible Stylesheet Language Transformations (XSLT)⁷ provided with the IP-XACT schema image. In addition, a DE or point tool may preserve information in the initial description for use outside of the scope of the IP-XACT specification.

1.4.1 Design environments

An IP-XACT–enabled DE shall

- Follow the parse validity requirements shown in [1.4](#).
- Create IP that is IP-XACT enabled.
- Modify any existing IP-XACT descriptions without losing any preexisting information. In particular, it shall preserve any vendor extension data included in the existing IP-XACT description.
- Be able to invoke IP-XACT–enabled generators with **apiType** of **none** (see [6.13.2](#)).

⁷Available from the W3C Web site at <http://www.w3.org/TR/xslt>.

An IP-XACT-TGI-enabled DE shall be considered as enabled as follows:

- For the base TGI, if it supports all generators utilizing the TGI_2014_BASE **apiType** (see 6.13.2).
- For the extended TGI, if it supports all generators utilizing the TGI_2014_BASE or TGI_2014_EXTENDED **apiTypes** (see 6.13.2).

1.4.2 Point tools

A point tool is a tool that has a particular, rather than a general, set of capabilities. In contrast to an IP-XACT-enabled DE (see 1.4.1), an IP-XACT-enabled point tool shall

- Follow the parse validity requirements shown in 1.4.
- Create IP that is IP-XACT enabled.
- Modify any existing IP-XACT descriptions without losing any preexisting information. In particular, it shall preserve any vendor extension data included in the existing IP-XACT description.

1.4.3 IPs

An IP-XACT-enabled IP shall

- Have an IP-XACT description that follows the description and semantic validity requirements shown in 1.4.
- Use IP-XACT-enabled generators for any generators associated with this IP.

XML descriptions compliant with IP-XACT shall provide a namespace reference to the `index.xsd` schema file, not to any of the other files in the release.

1.4.4 Generators

An IP-XACT-enabled generator shall

- Create IP that is IP-XACT enabled.
- Modify any existing IP-XACT descriptions without losing any preexisting information. In particular, it shall preserve any vendor extension data included in the existing IP-XACT description.
- Communicate with the DE that invoked it only through the IP-XACT TGI (see Annex F).

1.5 Conventions used

The conventions used throughout the document are included here.

IP-XACT is case-sensitive.

1.5.1 Visual cues (meta-syntax)

Bold shows required keywords and/or special characters, e.g., **addressSpace**. For the initial definitional use (per element), keywords are shown in **boldface-red text**, e.g., **bitsInLau** (see also 1.6).

Bold italics shows group names or data types, e.g., *nameGroup* or *boolean*. For definitions of types, see Annex D.

Courier shows examples, external command names, directories and files, etc., e.g., address 0x0 is on D[31:0].

1.5.2 Notational conventions

The keywords *required*, *shall*, *shall not*, *should*, *should not*, *recommended*, *may*, and *optional* in this standard are to be interpreted as described in IETF RFC-2119 [B5].⁸

1.5.3 Syntax examples

Any syntax examples shown in this standard are for information only and are intended only to illustrate the use of such syntax (see also [Annex I](#)).

1.5.4 Graphics used to document the schema

The W3C Web site⁹ specifies the XML schema language used to define the IP-XACT XML schemas. Normative details for compliance to the IP-XACT standard are contained in the schema files. Within this document, pictorial representations of the information in the schema files *illustrate* the structure of the schema and *define* any constraints of the standard. With the exception of visibility issues, the information in the figures and the schema files is intended to be identical (for a fully annotated version of the schema files; see IP-XACT Schema [B9]). Where the figures and schema are in conflict, the XML schema file shall take precedence.¹⁰

1.5.4.1 Elements and attributes

The *element* is the fundamental building block on which this standard is based. An element may be either a *leaf element*, which is a container for information, or a *branch element*, which may contain further branch elements or leaf elements.

A leaf or branch element may also contain *attributes*. Attributes are containers for information within the containing element.

1.5.4.2 Types

A *type* is a designation of the format for the contents of an element or attribute. There are two different styles of types that can be defined. A type may be assigned to a leaf element or an attribute. This type is called a ***simpleType*** and defines the format of data that may be stored in this container. A type may also be assigned to a branch element. This type is called a ***complexType*** and defines further elements and attributes contained in the branch element.

1.5.4.3 Groups

A group is a collection of elements or attributes, which allow the same collection of items to be referenced consistently in many places. There are two different types of groups that can be defined. A *group* is a combination of leaf or branch elements; an ***attributeGroup***, a simple list of attributes. The names assigned to either group have no representation in the resulting description.

⁸The number in brackets correspond to the numbers of the bibliography in [Annex A](#).

⁹Available from the W3C Web site at <http://www.w3.org/TR/REC-xmlschema/>.

¹⁰The graphics for this document have been generated by taking “screen-shots” of the various files as they are displayed in Altova’s XML environment XMLSpy[®]. XMLSpy is a registered trademark of Altova GmbH. This information is given for the convenience of users of this standard and does not constitute an endorsement by the IEEE of this product. Equivalent products may be used if they can be shown to lead to the same results.

1.5.4.4 Namespace

Each element, attribute, type, or group has a name, which is preceded by a namespace-prefix and separated from the name by a colon (:). For the examples in 1.5.4.5, **xyz** is used as the namespace-prefix for all of the items whereas this standard uses **ipxact**. Within the text of this standard, the namespace-prefix is not written when describing an item; it is shown only in examples.

1.5.4.5 Diagrams

The diagrams used throughout this standard graphically detail the organization of elements and attributes.

NOTE—For brevity, the **xml:id** attribute (see C.27) has been removed from all diagrams.¹¹

1.5.4.5.1 Elements and sequences

Figure 3 shows the sequence-compositor. At the left is a branch element, **element1**, with some descriptive text below. **element1** is connected to a sequence-compositor. The sequence-compositor defines the order the subelements appear in the branch element. **subElement1** shall appear first inside of **element1**. This is followed by **subElement2**, **subElement3**, **subElement4**, and **subElement5** before closing **element1**.

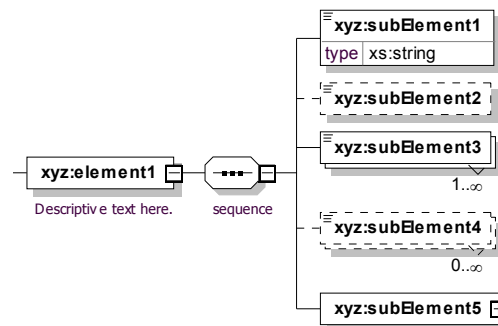


Figure 3—Sequence-compositor

- subElement1** is a mandatory element, as indicated by the solid line of the containing box. The type of the data contained in this element is set to **string**, and it has a default value of **ip-xact** if the element is present but left empty.
- subElement2** is an optional element, as indicated by the dashed-line of the containing box.
- subElement3** is an mandatory element that may appear multiple times, indicated by the double-solid line of the containing box. The number of times the element may appear is indicated by the range of the numbers listed below the element.
- subElement4** is an optional element that may appear multiple times, as indicated by the double-dashed line of the containing box. The number of times the element may appear is indicated by the range of the numbers listed below the element.
- subElement5** is an mandatory branch element that contains further elements inside, as indicated by the small plus sign (+) in the small box on the right.

¹¹Notes in text, tables, and figures are given for information only and do not contain requirements needed to implement the standard.

[Figure 4](#) shows variations of a sequence-compositor. **root1** is connected to an optional sequence-compositor, as indicated by the symbol being drawn with a dashed line. **element1** may appear first inside of **root1**; if it does, it shall be followed by **element2**. Each subelement is connected to a sequence-compositor.

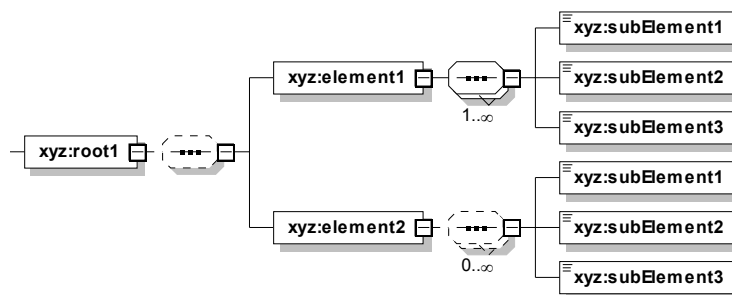


Figure 4—Sequence-compositor variations

- **element1** may contain one or more of the following sequences in the following order: **subElement1** and **subElement2** and **subElement3**. The number of times the sequence-compositor may appear is indicated by the range of the numbers listed below the symbol. If the range is greater than 1, the sequence-compositor symbol is drawn with double lines.
- **element2** is optional and may contain zero or more of the following sequences in the following order: **subElement1** and **subElement2** and **subElement3**. The number of times the sequence-compositor may appear is indicated by the range of the numbers listed below the symbol. If the range starts at 0 and the maximum is greater than 1, the sequence-compositor symbol is drawn with double-dashed lines.

1.5.4.5.2 Elements and choices

[Figure 5](#) shows the variations of the choice-compositor. **root** is connected to a choice-compositor. The choice-compositor specifies that one of the elements on the right side shall be chosen. **root** may contain one of the following: **element1**, **element2**, or **element3**. Each subelement is connected to a choice-compositor.

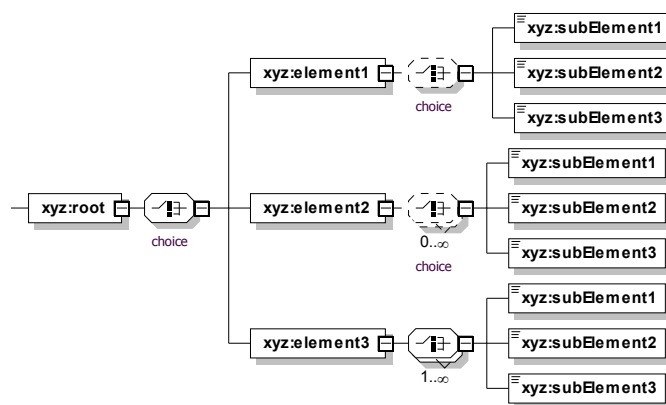


Figure 5—Choice-compositor variations

- a) **element1** may contain one of the following: **subElement1**, **subElement2**, or **subElement3**, as indicated by the symbol being drawn with a dashed line.
- b) **element2** may contain any (0 or more) of the following: **subElement1**, **subElement2**, or **subElement3** in any order. The number of times the choice-compositor may appear is indicated by the range of the numbers listed below the symbol. If the range starts at 0, the choice-compositor is drawn with dashed lines.

- c) **element3** may contain one or more of the following: **subElement1**, **subElement2**, or **subElement3** in any order. The number of times the choice-compositor may appear is indicated by the range of the numbers listed below the symbol. If the range is greater than 1, the choice-compositor is drawn with double lines.

1.5.4.5.3 Elements, attributes, groups, and attributeGroups

Figure 6 shows the use of attributes, groups, and attributeGroups. **element1** contains two attributes, shown in the tab-shaped box labeled *attributes*. **attribute1** is optional, as indicated by the dashed containing box, and is of type *integer* with a default value of 7 if the attribute is not present. **attribute2** is a required attribute, as indicated by the solid containing box, and is of type *boolean* with no default. The ordering in which **attribute1** and **attribute2** appear inside **element1** is irrelevant.

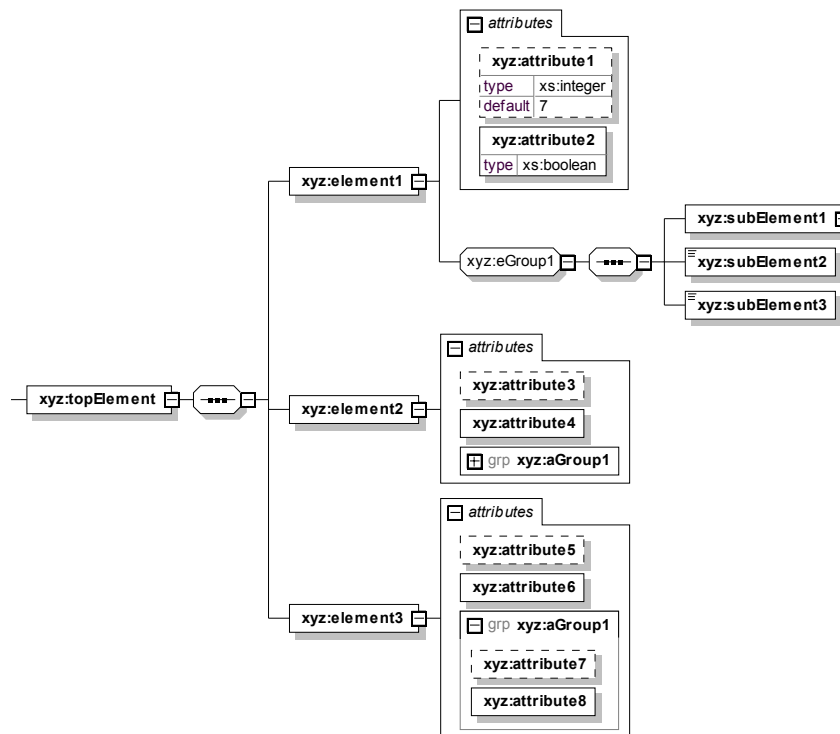


Figure 6—Attributes, groups, and attributeGroups

- a) **eGroup1** is an element group inside **element1**. This group contains three subelements, and the group symbol can be replaced by a solid line. The name of the group has no representation in the resulting output description. An element group can be optional, as indicated by a dashed outline (not shown), and it can also have a range, as indicated by numbers below the group symbol (not shown).
- b) **aGroup1** is an *attributeGroup* inside **element2** and **element3**. This *attributeGroup* contains two attributes, **attribute7** and **attribute8**. Inside **element2**, the *attributeGroup* is shown in its collapsed form, as indicated by the small plus sign (+) inside the small box. Inside **element3**, the *attributeGroup* is shown in its expanded form, as indicated by the small minus sign (–) inside the small box. **element2** contains four attributes: **attribute3**, **attribute4**, **attribute7**, and **attribute8**. **element3** also contains four attributes: **attribute5**, **attribute6**, **attribute7**, and **attribute8**. The name of the *attributeGroup* has no representation in the resulting description.

1.5.4.5.4 Wildcards

[Figure 7](#) shows the use of wildcards. A *wildcard* is depicted by the rounded box with the **any ##any** text. Wildcards indicate that any well-formed attribute or element may be inserted into the containing element.

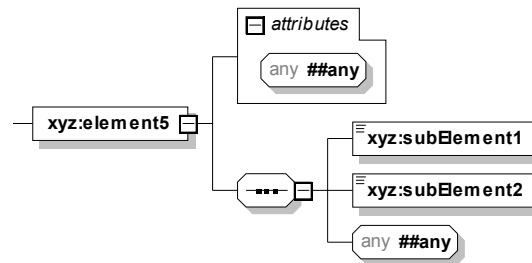


Figure 7—Wildcards

1.6 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).
- In the formal language definitions, syntactic keywords and tokens are shown in **boldface-red text**.

1.7 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 assumed or required for this standard.
- [Clause 3](#) defines terms, acronyms, and abbreviations used throughout the different specifications contained in this standard.
- [Clause 4](#) defines the interoperability use model.
- [Clause 5](#) defines the bus and abstraction definitions.
- [Clause 6](#) defines the component and interconnect models.
- [Clause 7](#) defines the designs and their connections.
- [Clause 8](#) defines the abstractor model between abstraction definitions.
- [Clause 9](#) defines the generator chain.
- [Clause 10](#) defines the design and generator chain configuration.
- [Clause 11](#) defines the catalog feature.
- [Clause 12](#) defines addressing.
- [Clause 13](#) defines data visibility.
- Annexes. Following Clause 13 are a series of annexes.

2. Normative references

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

eXtensible Markup Language (XML) schema specification, available from the W3C Web site at <http://www.w3.org/TR/xmlschema-0/>; <http://www.w3.org/TR/xmlschema-1/>; <http://www.w3.org/TR/xmlschema-2/>.

eXtensible Markup Language (XML) version 1.0 specification, available from the W3C Web site at <http://www.w3.org/TR/REC-xml/>.

eXtensible Stylesheet Language Transformations (XSLT) version 1.0 specification, available from the W3C Web site at <http://www.w3.org/TR/xslt>.

IEEE Std 1800™, Standard for SystemVerilog-Unified Hardware Design, Specification, and Verification Language.^{12, 13}

Simple Object Access Protocol (SOAP) version 1.2 specification, available from the W3C Web site at <http://www.w3.org/TR/soap12-part1/>.

Web Services Description Language (WSDL) version 1.1 specification, available from the W3C Web site at <http://www.w3.org/TR/wsdl>.

XML schema NameChar definition, available from the W3C Web site at <http://www.w3.org/TR/REC-xml/#NT-NameChar>.

XML schema NameStartChar definition, available from the W3C Web site at <http://www.w3.org/TR/REC-xml/#NT-NameStartChar>.

¹²IEEE publications are available from The Institute of Electrical and Electronics Engineers, Inc. (<http://standards.ieee.org/>).

¹³The IEEE standards or products referred to in this clause are trademarks of The Institute of Electrical and Electronics Engineers, Inc.

3. Definitions, acronyms, and abbreviations

For the purposes of this document, the following terms and definitions apply. *The IEEE Standards Dictionary Online* should be consulted for terms not defined in this clause.¹⁴

3.1 Definitions

abstraction definition: An object that describes a representation of a bus interface, including details of the ports this type of bus interface may have and the constraints that apply to these ports.

abstractor: A top-level IP-XACT element used to convert between two bus interfaces having different abstraction types and sharing the same bus type.

active interface: An interface that participates in the transactions.

ad hoc connection: A direct connection between two or more ports without the use of bus interfaces.

application programmers interface (API): A method for accessing design data and meta-data in a procedural way.

bridge: A mechanism to model the internal relationship between master interfaces and slave interfaces inside a component. Bridges explicitly describe the internal point-to-point connections between the component interfaces. A bridge can have multiple address spaces, supports memory mapping and remapping, and can have only direct interfaces. *Syn:* **bus bridge**.

broadcast: An interface connection type where one interface connects to multiple other interfaces.

bus: A collection of ports used to connect blocks connected to it involving both hardware and software protocols.

bus definition: An object that describes the type properties for a bus, such as the maximum masters allowed or if one bus expands upon the definition of another.

bus interface: The interface of an intellectual property (IP) to an interconnection. Components are connected together by linking the bus interfaces together. The three different classes of bus interfaces, master, slave, and system, have two flavors: direct and mirrored.

catalog: An object that defines a mapping from IP-XACT Vendor Library Name Version (VLNV) to the physical file in which the IP-XACT object is defined.

channel: A mechanism to model the internal relationship between mirrored-slave interfaces and mirrored-master interfaces inside a component. A channel can also represent a simple wiring interconnection or a more complex structure, such as a bus. A channel can have only one address space. Channel interfaces are always mirrored interfaces. A channel supports memory mapping and remapping.

component: An object containing IP-XACT meta-data for an intellectual property (IP). Components are used to describe computing cores, peripherals, and bussing structures. *Syn:* **component description**.

configurable element: An element in an IP-XACT description that can be set to a new value by a user, generator, or dependency equation. This includes all elements with a resolve attribute of user or generated.

¹⁴The *IEEE Standards Dictionary Online* subscription is available at
http://www.ieee.org/portal/innovate/products/standard/standards_dictionary.html.

configurable intellectual property (IP): IP that contains configurable elements and/or an IP-specific generator capable of creating new components from the configured component and updating the design with the new version of the component. *Syn:* **configurable component**.

configuration: Replacing the parameter values with values specified in `configurableElementValue` elements. This includes evaluating each `IsPresent` element value and then treating the containing element as not present if the expression evaluates to 0. Certain semantic rules apply only after this processing.

NOTE—See [C.10](#).

constraint: A limitation on a part of the system that needs to be satisfied for the system to be correct. Timing constraints are often specified on ports, requiring that during a given clock cycle the value of the port becomes stable in a certain time period and remains stable for a certain time period relative to a particular clock edge.

constraint set: Constraints defined in groups to associate different constraints with different views of the component.

design: An IP-XACT description of a system or subsystem listing its components and the connections between these components.

design configuration: A design description that contains non-essential ancillary information for generators, the active or current view selected for instances in the design, and configurable information defined in vendor extensions. It references a design description, can specify a view for the component instances and abstractors for each interconnection, and can configure generator chains. *Syn:* **configuration**.

design database: Working storage for both meta-data and component information that helps create and verify systems and subsystems.

design environment (DE): A software tool or set of tools that manages the access to IP-XACT objects, coordinates the execution of generator chains, and provides an application programmers interface (API) between generators and IP-XACT objects.

electronic design intellectual property (IP): In the electronic design community, a reusable collection of design specifications that represent the behavior, properties, and/or description of the design in various media. The name IP is partially derived from the common practice of considering a collection of this type to be the intellectual property of one party. Both hardware and software collections are encompassed by this term. IP utilized in the context of a system on chip (SoC) design or design flow may include specifications; design models; design implementation descriptions; verification coordinators, stimulus generators, checkers, and assertion/constraint descriptions; soft design objects (such as embedded software and real-time operating systems); and design and verification flow information and scripts. IP-XACT distinguishes between fixed IP and configurable IP.

endianness: The system of ordering bytes in computer memory where big endian is the most significant byte at the lowest memory address and little endian is the least significant byte at the lowest memory address.

eXtensible Markup Language (XML): A simple, very flexible text format derived from Standard Generalized Markup Language (SGML).

NOTE—See ISO/IEC 8879 [\[B15\]](#).

eXtensible Stylesheet Language Transformations (XSLT): A language for transforming eXtensible Markup Language (XML) documents into other XML documents.

fixed intellectual property (IP): IP that has no elements that are configured by the design environment (DE) or set by industry de facto tools.

generator: An external application that can communicate with a design environment (DE) using the tight generator interface (TGI) to complete a specific task. Common examples include documentation generation, design data manipulation (adding/removing/updating IP-XACT elements), and flow automation.

generator chain: A hierarchical list of generators used to define the order for executing generators. A design flow can be represented by a generator chain.

generator group: A symbolic name assigned to a generator to enable generator selection.

generator invocation: A method of running an application at a defined phase in the generator group with a given number of elements.

hardware description language (HDL) path: A hierarchical path to a memory mapped object (register or memory) within a design. HDL paths facilitate back-door accesses. In IP-XACT, a view-specific HDL path can be defined for each memory mapped object. HDL paths are stored in a distributed manner across the memory map hierarchy using an accessHandle.

NOTE—See [C.1](#).

hierarchical component: A component that has one or more **views** that reference IP-XACT design or design configuration descriptions.

hierarchical family of bus interfaces: A hierarchical family of bus interfaces is a set of bus interfaces composed of a hierarchical bus interface and all its hierarchical descendants.

hierarchical family of components: A hierarchical family of components is a component and all its hierarchical descendants.

indirect interface: The access mechanism for an indirectly accessible memory map. This includes defining indirect address and data fields along with addressing information.

indirectly accessible memory map: A memory map whose contents cannot be accessed at a fixed address via a bus interface and, instead, are indirectly accessible via indirect address and data fields. A bus interface accesses an indirectly accessible memory map via reads and writes to indirect address and data fields.

initiative: An abstract description of port modes (requires, provides, both, or phantom) used for transaction-level modeling (TLM).

intellectual property (IP) integrator: A party in the design process who receives IP and subsystems and combines them into a larger system.

intellectual property (IP) platform architect: The creator of platform-based architectures.

intellectual property (IP) provider: The creator and supplier of IP.

intellectual property (IP) repository: The database of IP.

interconnection: The point-to-point connection between two or more bus interfaces.

leaf component: A component that does not contain any views that reference IP-XACT design or design configuration descriptions.

master interface: The bus interface that initiates a transaction (like a read or write) on a bus.

memory map: A block of memory in a component (which may be accessible through a slave interface).

meta-data: A tool-interpretable way of describing information, such as the design history, locality, association, configuration options, and integration requirements of an object as well as constraints against an object.

mirror interface: An interface that has the same (or similar) ports as its related direct bus interface, but whose port directions are reversed. Therefore, a port that is an input on a direct bus interface would be an output in the matching mirror interface.

monitor interface: An interface used in verification that is not a master, slave, or system interface.

multi-layer buses: Buses that have to be modeled as component bridges with direct interfaces or as a hierarchical component.

objects: XML descriptions of the following types: catalogs, components, designs, busDefinitions, abstractionDefinitions, abstractors, designConfigurations, and generatorChains. To be able to be uniquely referenced, each object has a unique identifier called its Vendor Library Name Version (VLNV).

opaque bridge: A bus interconnect component that may modify the address space of a master bus interface of one bus type to the memory map of a slave bus interface of another bus type and does not allow direct access to any components residing on that address space. An opaque bridge has the opaque attribute equal to true.

phantom port: A port that exists only in an IP-XACT component; it does not exist on any hardware description language (HDL) or transaction-level modeling (TLM) component.

phase number: The sequence in which generators should be fired.

platform: Architectural (sub)system framework.

platform consumer: The user/group that builds a system on chip (SoC) based on a particular platform.

platform provider: The user/group that develops and delivers platforms to platform consumers.

platform rules: Rules that define how components interface with a specific platform.

port: The interface items of a component. These interface items allow dynamic exchange of information. Connections between ports may be specified by using ad hoc connections or by including them in bus interfaces connected together by interconnections.

schema: A means for defining the structure, content, and semantics of eXtensible Markup Language (XML) documents.

segment: A portion of an addressSpace, defined with an address offset and range.

semantic consistency rules (SCRs): Additional rules applied to an eXtensible Markup Language (XML) description that cannot be expressed in the schema. Typically, these are rules between elements in multiple XML descriptions.

slave interface: The bus interface that terminates or consumes a transaction initiated by a master interface. Slave interfaces often contain information about the registers accessible through the slave interface.

system interface: An interface that is neither a master nor slave interface and allows specialized (or non-standard) connections to a bus (e.g., clock).

tight generator interface (TGI): An interface used to manipulate values of elements and attributes for IP-XACT-compliant XML.

transactional port: A port that has an initiative and a kind or a type definition (which can specify the data type of the port). Transactional ports are used for high-level modeling.

transaction-level modeling (TLM): An abstraction level higher than register transfer level (RTL), used for specifying, simulating, verifying, implementing, and evaluating system on chip (SoC) designs.

transparent bridge: A bus interconnect component that modifies the address space of a master bus interface of one bus type to the memory map of a slave bus interface of another bus type with directly addressable access to any components residing on that address space. A transparent bridge has the opaque attribute equal to false.

use model: A process method of working with a tool.

user interface: Methods of interacting between a tool and its user.

validation: The process of proving the correctness of construction of a set of components.

Vendor Library Name Version (VLNV): The unique identifier assigned to each top-level IP-XACT object and specified in each eXtensible Markup Language (XML) file.

verification: The process of proving the behavior of a set of connected components.

verification intellectual property (VIP): Components included in a design for verification purposes.

view: An implementation of a component. A component may have multiple views, each with its own function in the design flow.

virtual register: A collection of fields, overlaid on top of a memory, usually in an array. The semantics and layout of virtual registers come from an agreement between the software and hardware. Virtual registers are modeled in IP-XACT by creating register instances within addressBlocks with usage memory. The child elements of a virtual register are restricted to a subset of the overall register element.

NOTE—See [6.11.2](#).

virtual register file: A grouping of virtual registers. A virtual register file is modeled using the registerFile element in an addressBlock element with usage of memory. The children of virtual register files shall be virtual.

NOTE—See [6.11.6](#).

wire connections: Connections that connect wire ports.

wire port: A port that describes binary values or an array of binary values. Wire ports can have a direction: in, out, or inout.

3.2 Acronyms and abbreviations

API	application programmers interface
DE	design environment
EDA	electronic design automation
HDL	hardware description language
HTTP	Hypertext Transfer Protocol
IP	(electronic design) intellectual property
LAU	least addressable unit (of memory)
RAM	random access memory
ROM	read-only memory
RTL	register transfer level (design)
SCR	semantic consistency rule
SOAP	Simple Object Access Protocol
SoC	system on chip
TGI	tight generator interface
TLM	transaction-level modeling
VIP	verification IP
VLNV	Vendor Library Name Version
WSDL	Web Services Description Language
XML	eXtensible Markup Language
XSLT	eXtensible Stylesheet Language Transformations

4. Interoperability use model

To introduce the use model for the IP-XACT specifications, it is first necessary to identify specific roles and responsibilities within the model and then relate these to how the IP-XACT specifications impact their interactions. All or some of the roles can be mixed within a single organization, e.g., some EDA providers are also providing IP, a component IP provider can also be a platform provider, and an IP system design provider may also be a consumer.

4.1 Roles and responsibilities

For this standard, the roles and responsibilities are restricted to the scope of IP-XACT HDL and TLM system design.

4.1.1 Component IP provider

A component IP provider is a person, group, or company creating IP components or subsystems for integration into a SoC design. These IPs can be hardware components (e.g., processors, memories, buses), verification components, and/or hardware-dependent software elements. They may be provided as source files or in a compiled form (i.e., simulation model). An IP is usually provided with a functional description, a timing description, some implementation or verification constraints, and some parameters to characterize (or configure) the IP. All these types of characterization data may be described as meta-data compliant with the IP-XACT schema. Elements not already provided in the base schema can be provided using namespace extensibility mechanisms of the specification.

The IP provider can use one or more EDA tools to create/refine/debug IP. During this process, the IP provider may export and re-import its design from one environment to another. The IP-XACT IP descriptions need to enable this exchange for component IP.

At some point, this IP can be transferred to customers, partners, and external EDA tool suppliers by using IP-XACT-compliant XML. IP can be characterized as *fixed IP* or *configurable IP*.

4.1.2 SoC design IP provider

A SoC design IP provider is a person, group, or company that integrates and validates IP provided by one or more IP providers to build system platforms, which are complete and validated systems or subsystems. Like the IP provider, the platform provider can use EDA tools to create/refine/debug its platform, but at some point the IP needs to be exchanged with others (e.g., customers, partners, other EDA tools). To do so, the platform IP has to be expressed in the IP-XACT-specified format as a hierarchical component.

4.1.3 SoC design IP consumer

A SoC design IP consumer is a person, group, or company that configures and generates system applications based on platforms supplied by SoC design IP providers. These platforms are complete system designs or subsystems. Like the platform provider, the platform consumer can use EDA tools to create/refine/debug its system application and/or configure the design architecture. To do so, the EDA tool needs to support any platform IP expressed in the IP-XACT-specified format.

4.1.4 Design tool supplier

A design tool supplier is a group or company that provides tools to verify and/or implement an IP or platform IP. There are three major tools (which could be combined) provided in a system flow:

- Platform builder (or *system design environment*) tools: these help to assemble a platform with some automation (e.g., automatic generation of interconnect).

- Verification point tools: these handle functional and timing simulation, verification, analysis, debugging, co-simulation, co-verification, and acceleration.
- Implementation point tools: these handle synthesizing, floor-planing, place, routing, etc.

The EDA provider needs to be able to import IP-XACT component or system IP libraries from multiple sources and export them in the same format.

Further, IP-XACT EDA tools need to recognize, associate, and launch generators that may be provided by a generator or IP provider in support of configurable IP bundles. The imported IP might need to be created and/or modified by the tool and then exported back (e.g., to be exchanged with other EDA vendor tools) to satisfy the customer design flow.

To further support any generators supplied with IP bundles, the IP-XACT DE tools need to be able to recognize and interface with generator-wrapped point tools. These may be provided by another EDA provider or by the IP designer/consumer as part of a company's internal design and verification flow. In general, these support specialized design-automation features, such as architectural-rule checking.

4.2 IP-XACT IP exchange flows

This subclause describes a typical IP exchange flow that the IP-XACT specifications technically support between the roles defined in 4.1. By way of example, the following specific exchange flow can benefit from use of the IP-XACT specification:

The component IP provider generates an IP-XACT XML package and sends it to a SoC design tool (EDA tool supplier) or directly to a platform (i.e., SoC design IP) provider. The EDA tool supplier imports IP-XACT XML IP and generates platform IP and/or updates (configures) the IP components. The platform provider generates a configurable platform IP and exports it in IP-XACT XML format, which the end user imports to build system applications. The platform provider can also generate its own platform IP into IP-XACT format and send it to the EDA provider.

Although many different possible IP exchange flows exist, from the user's viewpoint, there are three main use models, as follows:

- IP (component or SoC design) provider use model
- Generator (IP provider and design tool) provider use model
- SoC design tool provider use model

4.2.1 Component or SoC design IP provider use model

The IP provider (a hardware component IP designer or platform IP architect) can use IP-XACT to package IP in a standard and reusable format. The first step consists in creating an IP-XACT XML package (XML plus any IP views) to export the IP database in a valid format. To express this IP as an IP-XACT IP, the IP provider needs to parse the entire design description tree (which is composed of files of different types: HDL source files, data sheets, interfaces, parameters, etc.) and package it into an IP-XACT XML format. This can be a manual step (by directly editing IP-XACT-compliant XML) or an automated one (using scripts to generate schema-compliant IP-XACT XML).

Once the IP has been packaged in an IP-XACT format, the IP provider can use a SoC design tool to write/debug/simulate/implement the IP.

4.2.2 Generator provider use model

The author of a generator expects to interact with the SoC design tool through a fixed interface during well-defined times in the design life cycle: when components are instantiated or modified or when a generator chain is started.

Generators are used within the SoC design tool to extend its capabilities: wrapping a point tool, e.g., a simulator; wiring up IP within the design; or checking the design is correct or maybe modifying the design. Many of these features may be supplied by the IP author and handled by generators embedded in the IP itself.

Consequently, there are at least two groups of generator providers: IP vendors who supply generators that are written specifically to support their IP and generic generator authors who wish to extend the features available within the SoC design tool. This latter group will be mainly SoC design tool vendors at first, but will also come to include third-party generator vendors.

4.2.3 System design tool provider use model

The system design tool takes IP-XACT components and designs as input, configures them, and loads them into its own database format. Then it can automate some tasks, such as creating the platform, generating the component interconnect and bus fabric, and generating or updating the IP-XACT IP as an output (by providing new or updated XML with the attached information, e.g., new source files, parameters, documentation).

Customer design flows are usually composed of a chain of different tools from the same or different EDA vendors (e.g., when an EDA provider is not providing the entire tool chain to cover all the user flow or the customer is selecting the best-in-class point tools). To address this requirement, the EDA vendor providing an IP-XACT-enabled tool needs to read and produce the IP-XACT-specified format and then utilize and implement the interfaces defined by IP-XACT documents. In this use model, each SoC design tool uses its own generators (possibly utilizing the IP-XACT TGI) to build and update its internal meta-data state and export to an IP-XACT format. Then the IP-XACT description can be imported by another IP-XACT-enabled EDA tool.

5. Interface definition descriptions

5.1 Definition descriptions

In IP-XACT, a group of ports that together perform a function are described by a set of elements and attributes split across two descriptions: a bus definition and an abstraction definition. These two descriptions are referenced by components or abstractors in their bus or abstractor interfaces.

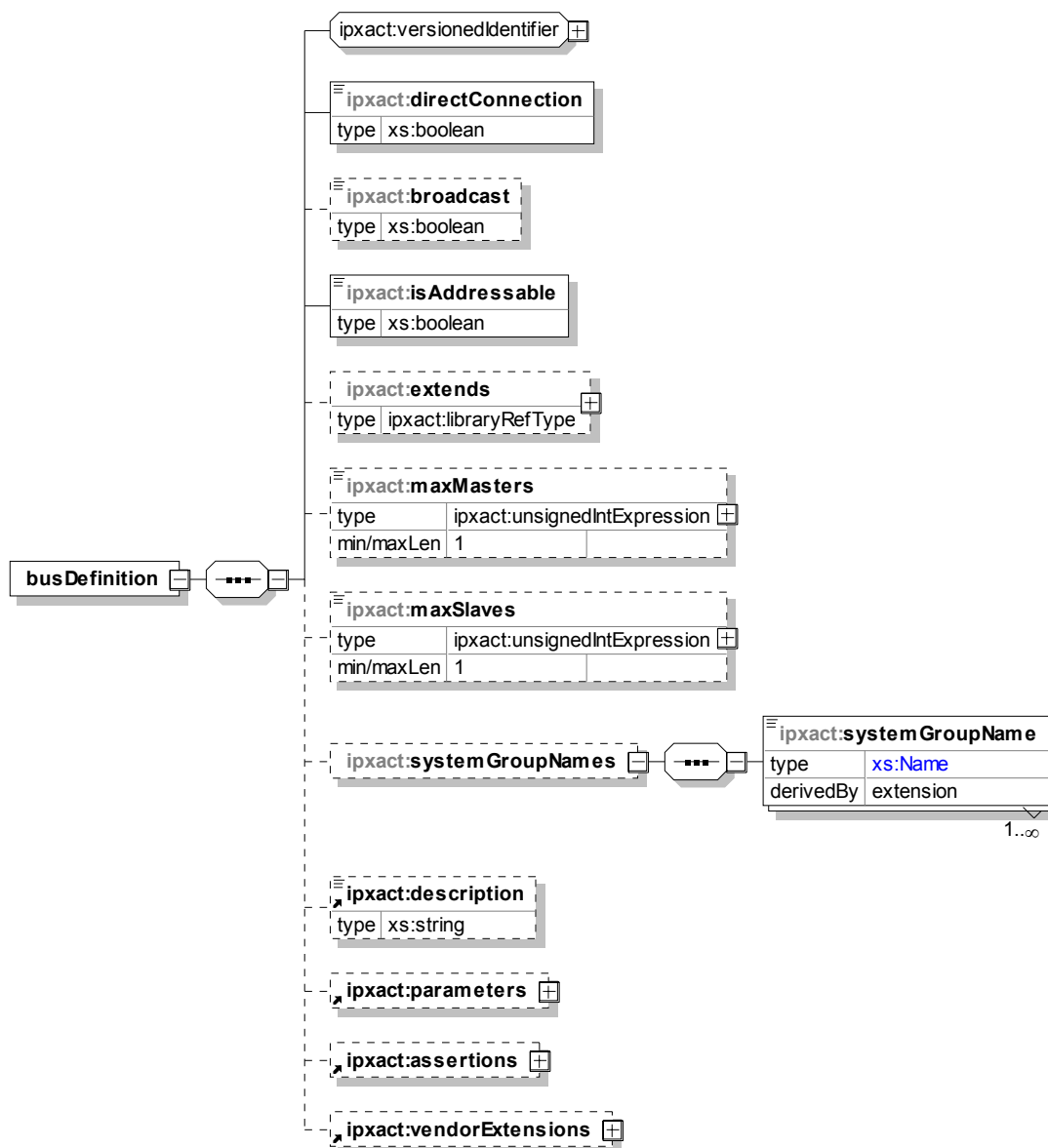
The *bus definition* description contains the high-level attributes of the interface, including items such as the connection method and indication of addressing.

The *abstraction definition* contains the low-level attributes of the interface, including items such as the name, direction, and width of the ports. This is a list of logical ports that may appear on a bus interface for that bus type. Multiple abstractions definitions can be associated with a single bus definition. See [6.5](#).

5.2 Bus definition

5.2.1 Schema

The following schema details the information contained in the **busDefinition** element, which is one of the top-level elements in the IP-XACT specification used to describe the high-level aspects of a bus.



5.2.2 Description

The top-level **busDefinition** element describes the high-level aspects of a bus or interconnect. It contains the following elements and attributes:

- The **versionedIdentifier** group provides a unique identifier; it consists of four subelements for a top-level IP-XACT element. See [C.25](#).
- directConnection** (mandatory; type: **boolean**) specifies what connections are allowed. A value of **true** specifies these interfaces may be connected in a direct master-to-slave fashion. A value of **false**

indicates only non-mirror to mirror type connections are allowed (i.e., master–mirroredMaster, slave–mirroredSlave, or system–mirroredSystem).

- c) **broadcast** (optional; type: *boolean*; default: **false**) indicates this bus definition supports *broadcast mode*, i.e., bus interfaces using this definition support one-to-many interface connections.
- d) **isAddressable** (mandatory; type: *boolean*) specifies the bus has addressing information. A value of **true** specifies these interfaces contain addressing information and a memory map can be traced through this interface. A value of **false** indicates these interfaces do not contain any traceable addressing information. See also [6.3](#).
- e) **extends** (optional; type: *libraryRefType* (see [C.11](#))) specifies whether this definition is an extension from another bus definition. See also [5.11](#).
- f) **maxMasters** (optional; type: *unsignedIntExpression* (see [C.3.7](#))) specifies the maximum number of masters that can appear in a **channel** element containing bus interfaces using this bus definition. If the **maxMasters** element is not present, the number of allowed masters is unbounded.
- g) **maxSlaves** (optional; type: *unsignedIntExpression* (see [C.3.7](#))) specifies the maximum number of slaves that can appear in a **channel** element containing bus interfaces using this bus definition. If the **maxSlaves** element is not present, the number of allowed slaves is unbounded.
- h) **systemGroupNames** (optional) defines an unbounded list of **systemGroupName** (mandatory; type: *Name*) elements, which in turn define the possible group names to be used under an **onSystem** element in an abstraction definition. The definition of the group names in the bus definition allows multiple abstraction definitions to indicate which system interfaces match each other. The **systemGroupName** shall be unique with the containing **systemGroupNames** element.
- i) **description** (optional; type: *string*) allows a textual description of the interface.
- j) **parameters** (optional) describes any **parameter** that can be used to configure or hold information related to this bus definition. See [C.18](#).
- k) **assertions** (optional) contains a list of expressions defining the allowed parameter values. See [C.2](#).
- l) **vendorExtensions** (optional) contains any extra vendor-specific data related to the interface. See [C.24](#).

See also [SCR 1.3](#), [SCR 1.10](#), [SCR 1.12](#), [SCR 2.17](#), [SCR 3.18](#), and [SCR 6.16](#).

5.3 Abstraction definition

5.3.1 Schema

The following schema details the information contained in the **abstractionDefinition** element, which is one of the top-level elements in the IP-XACT specification used to describe the low-level aspects of a bus.



5.3.2 Description

The **abstractionDefinition** element describes the low-level aspects of a bus or interconnect. It contains the following elements and attributes:

- The **versionedIdentifier** group provides a unique identifier; it consists of four subelements for a top-level IP-XACT element. See [C.25](#).
- busType** (mandatory; type: **libraryRefType** (see [C.11](#))) specifies the bus definition that this abstraction defines. See also [5.11](#).
- extends** (optional; type: **libraryRefType** (see [C.11](#))) specifies whether this definition is an extension from another abstraction definition. The extending abstraction definition may change the definition of logical ports, add new ports, or mark existing logical ports illegal (to disallow their use). See also [5.11](#).
- ports** (mandatory) is a list of logical ports; see [5.4](#).
- description** (optional; type: **string**) allows a textual description of the interface.
- parameters** (optional) describes any **parameter** that can be used to configure or hold information related to this bus definition. See [C.18](#).
- assertions** (optional) contains a list of expressions defining the allowed parameter values. See [C.2](#).
- vendorExtensions** (optional) contains any extra vendor-specific data related to the interface. See [C.24](#).

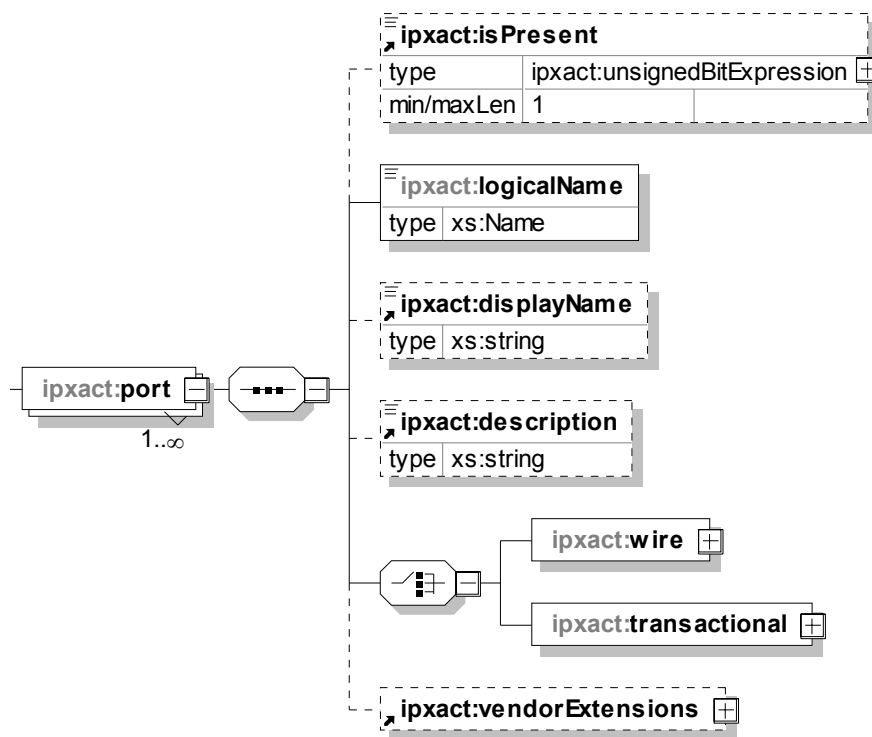
The **abstractionDefinition** element contains a list of logical ports that define a representation of the bus type to which it refers. A port can be a wire port (see 5.7) or a transactional port (see 5.9). A *wire port* carries logic information or an array of logic information. A *transactional port* carries information that is represented at a higher level of abstraction.

See also [SCR 1.10](#), [SCR 1.12](#), [SCR 1.14](#), [SCR 3.1](#), [SCR 3.15](#), [SCR 3.16](#), and [SCR 6.11](#).

5.4 Ports

5.4.1 Schema

The following schema details the information contained in the **ports** element, which appears as part of the **abstractionDefinition** element within an abstraction definition. This is different from the **ports** element that appears as part of the **model** element within components.



5.4.2 Description

A **port** element defines the logical port information for the containing abstraction definition. It contains the following elements:

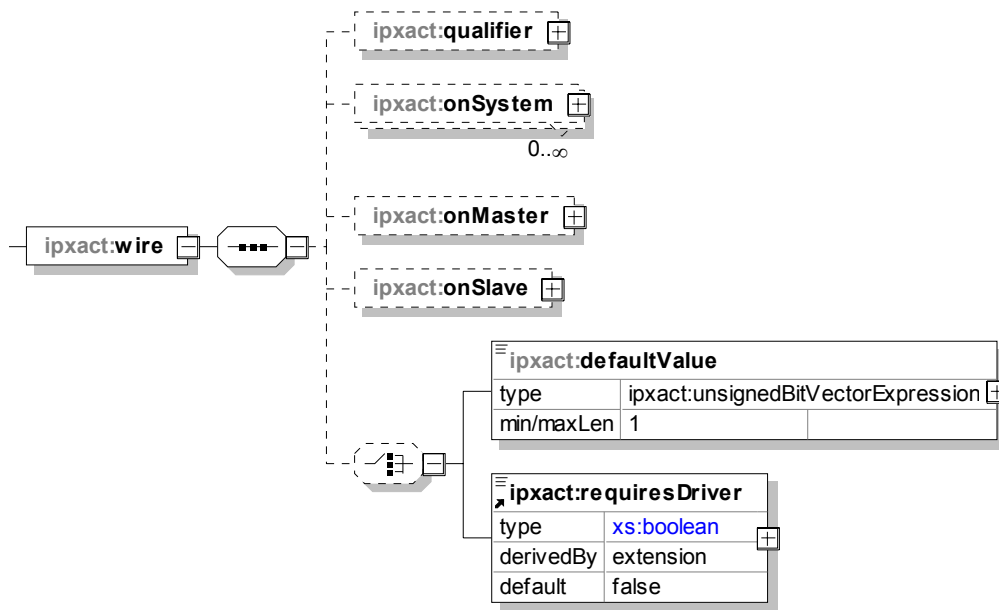
- The **isPresent** (optional; type: *unsignedBitExpression* (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.
- logicalName** (mandatory; type: *Name*) gives a name to the logical port that can be used later in component description when the mapping is done from a logical abstraction definition port to the components physical port. The **logicalName** shall be unique within the **abstractionDefinition**.
- displayName** (optional; type: *string*) allows a short descriptive text to be associated with the port.
- description** (optional; type: *string*) allows a textual description of the port.

- e) Each **port** also requires a **wire** element or a **transactional** element to further describe the details about this port. See [5.5](#) or [5.9](#), respectively. A **wire** style port is a port that carries logic values or an array of logic values. A **transactional** style port is a port that carries any other type of information, typically used for TLM.
- f) **vendorExtensions** (optional) contains any extra vendor-specific data related to the port. See [C.24](#).

5.5 Wire ports

5.5.1 Schema

The following schema details the information contained in the **wire** element, which may appear as part of the **port** element within an abstraction definition (**abstractionDefinition/ports/port**).



5.5.2 Description

A **wire** element represents a port that carries logic values or an array of logic values. This logical wire port may provide optional constraints for a wire port, to which it is mapped inside a component or abstractor's **busInterface**. It contains the following elements and attributes:

- a) **qualifier** (optional) indicates which type of information this wire port carries. See [5.6](#).
- b) **onSystem** (optional) defines constraints, e.g., timing constraints, for this wire port if it is present in a system bus interface with a matching group name.
 - 1) The **group** (mandatory) attribute indicates the group name for the wire port. It distinguishes between different sets of system interfaces. Usually, all the arbiter ports are processed together, or all the clock or reset ports are processed together. So, this is really a mechanism to specify any sort of non-standard bus interface capabilities for the interconnect. The type of this element is *Name*.
 - 2) The group **wirePort** specifies what elements are used in this port. See [5.7](#).
- c) **onMaster** (optional) defines constraints for this wire port when present in a master bus interface. The group **wirePort** specifies what elements are used in this port. See [5.7](#).

- d) **onSlave** (optional) defines constraints for this wire port when present in a slave bus interface. The group **wirePort** specifies what elements are used in this port. See [5.7](#).
- e) A wire may also contain one of the following elements:
 - 1) **defaultValue** (type: *unsignedBitVectorExpression* (see [C.3.6](#))) contains the default logic value for this wire port. This value is applied when the **busInterface** is connected, this logical port is not connected, and there is no ad hoc connection to the corresponding physical port.
 - 2) **requiresDriver** (type: *boolean*; default: **false**) specifies whether the port requires a driver when used in a completed design. If this element is not present, its effective value is **false**, indicating this port does not require a driver. When set to **true**, the attribute **driverType** further qualifies what driver type is required: **any** (any logic signal or value), **clock** (a repeating type waveform), or **singleshot** (a non-repeating type waveform). If **driverType** is not present, its effective value is **any**.

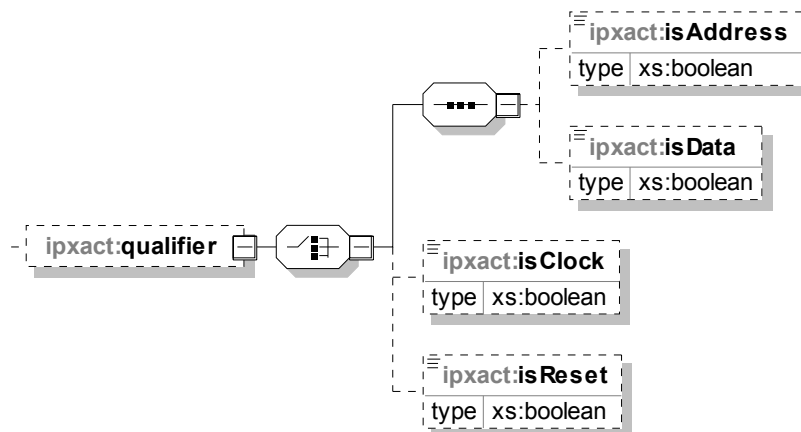
NOTE—The **onMaster**, **onSlave**, and **onSystem** elements associated with each logical port provide optional constraints. If any of these is missing, there are no constraints for how the port appears on interfaces with that mode (master, system, or slave). A port may appear in any system interface group unless its presence is marked as illegal for that group. The abstraction definition author has the choice of how far to constrain the definitions. Generally speaking, more constraints in the definitions reduce implementation flexibility for whoever is creating IP with interface that conforms to the abstraction definition.

See also [SCR 6.12](#) and [SCR 6.14](#).

5.6 Qualifiers

5.6.1 Schema

The following schema details the information contained in the **qualifier** element, which may appear as part of the **wire** element within an abstraction definition (**abstractionDefinition/ports/port/wire**).



5.6.2 Description

The **qualifier** element indicates which type of information a wire port carries. It contains the following elements:

- a) **isAddress** (optional; type: *boolean*), when **true**, specifies the port contains address information. This **qualifier** may be paired with the **isData** element (e.g., used with serial protocols). See also [Clause 12](#).
- b) **isData** (optional; type: *boolean*), when **true**, specifies the port contains data information. This data resides in registers defined in the memory map referenced by the interface. The width defined by the

port on each side of the two connected bus interfaces can be used to determine which portions of the data may be lost or gained (tied off to defaults) during transfers if the two widths do not match. This **qualifier** may be paired with the **isAddress** element (e.g., used with serial protocols). See also [Clause 12](#).

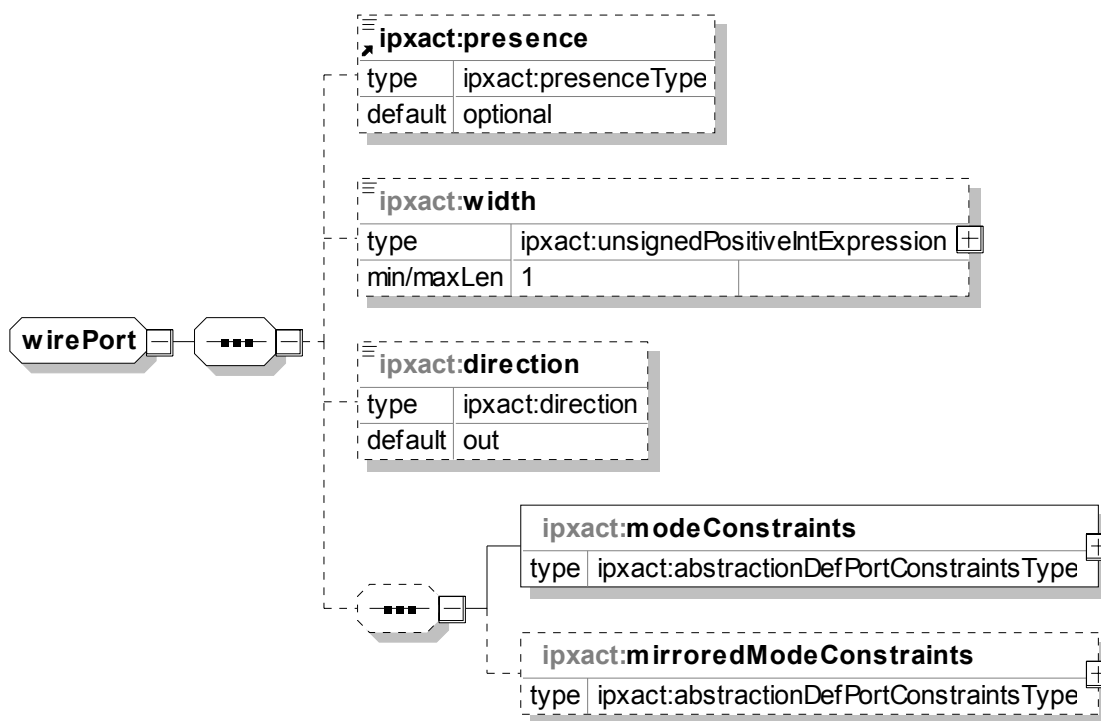
- c) **isClock** (optional; type: *boolean*), when **true**, specifies this port is a clock for this bus interface, i.e., it provides a repeating pattern that the interface uses to implement the protocol. No method of processing is implied with this tag. This tag shall be applied only to pure clock ports. This **qualifier** shall not be combined with other qualifiers.
- d) **isReset** (optional; type: *boolean*), when **true**, specifies this port is a reset for this bus interface, i.e., it provides the necessary input to put the interface into a known state. No method of processing is implied with this tag. This tag should be applied only to pure reset ports. This **qualifier** shall not be combined with other qualifiers.

See also [SCR 6.16](#), [SCR 9.1](#), [SCR 9.2](#), and [SCR 12.8](#).

5.7 Wire port group

5.7.1 Schema

The following schema details the information contained in the *wirePort* group, which may appear as part of the **onSystem**, **onMaster**, or **onSlave** element within a **wire** element within an abstraction definition (**abstractionDefinition/ports/port/wire/onmode**).



5.7.2 Description

The **wirePort** group specifies what elements are used in a **wire** port. It contains the following elements:

- presence** (optional; default: **optional**) provides the capability to require or forbid a port from appearing in a **busInterface**. The three possible values are **illegal**, **required**, or **optional**. If this element is not present, its effective value is **optional**.
- width** (optional; type: *unsignedPositiveIntExpression* (see C.3.9)) represents the number of logical bits that are required to represent this port. When mapping to this logical port in a **busInterface/portmap**, the numbering shall start from 0 to width-1. If **width** is not specified, the component shall define the number of bits in this port, but the logical portmap numbering shall still start at 0. If necessary, logical bit 0 shall be the least significant bit.
- direction** (optional; default: **out**) restricts the direction of the port relative to the non-mirrored interface (see 6.2). The three possible values are **in**, **out**, or **inout**.
- Each **wirePort** group can also have a sequence of **modeConstraints** and **mirroredModeConstraints** specifying the default synthesis constraints associated with this logical port. The **modeConstraints** apply to this port if it appears in a non-mirrored *mode* bus interface (see 5.8). Any **mirroredModeConstraints** apply to this port if it appears in a mirrored-*mode* bus interface (see 5.8).

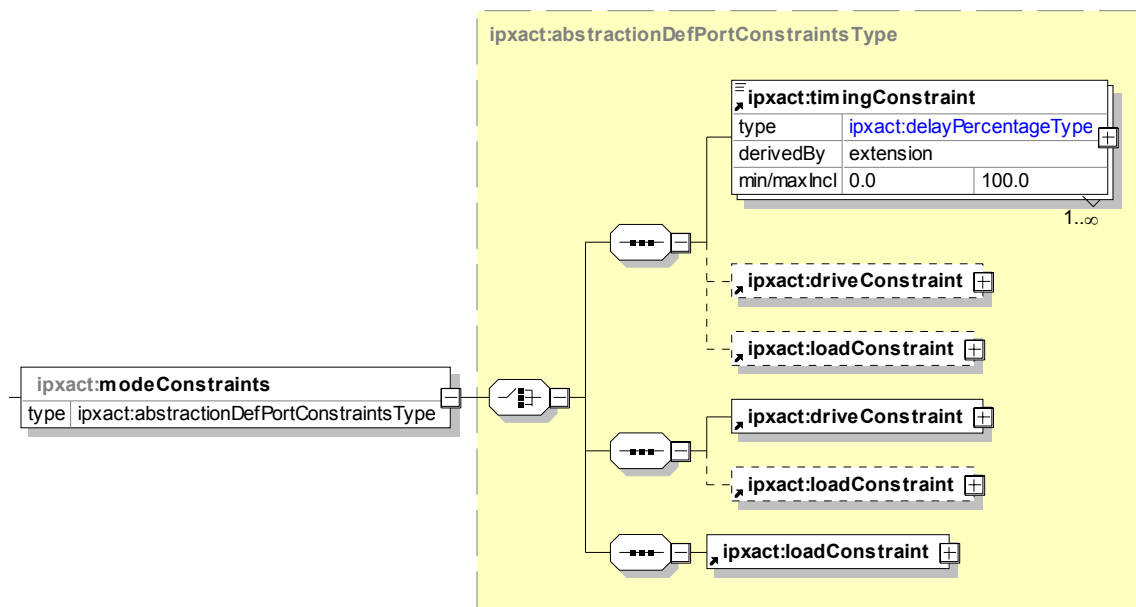
If **mirroredModeConstraints** are not specified, the **modeConstraints** also apply to this port in a mirrored-*mode* bus interface.

See also [SCR 6.5](#), [SCR 6.6](#), [SCR 6.7](#), and [SCR 6.17](#).

5.8 Wire port *mode* (and mirrored mode) constraints

5.8.1 Schema

The following schema defines the information contained in the **modeConstraints** and **mirroredModeConstraints** elements, which may appear within an **onMaster**, **onSlave**, or **onSystem** element within an abstraction definition (**abstractionDefinition/ports/port/wire/onmode**).



5.8.2 Description

The *abstractionDefPortConstraintsType* type definition (used to define **modeConstraints** and **mirroredModeConstraints**) defines any default implementation constraints associated with the containing wire port of the abstraction definition. It contains one or more of the following elements:

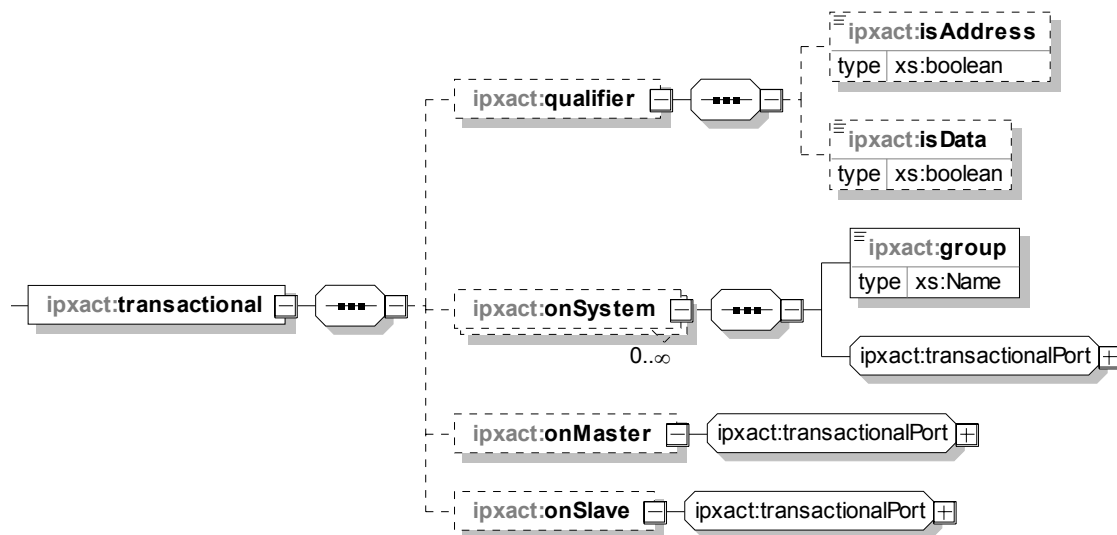
- timingConstraint** (optional) element defines a technology-independent timing constraint associated with the containing wire port. See 6.12.14.
- driveConstraint** (optional) element defines a technology-independent drive constraint associated with the containing wire port. See 6.12.14.
- loadConstraint** (optional) element defines a technology-independent load constraint associated with the containing wire port. See 6.12.14.

The constraints contained within the **modeConstraints** element are applied to the corresponding physical ports in a component only when the physical port does not have any constraints defined within its own port element and no standard design constraint (SDC) file is associated with the component. For example, if it appears inside an **onMaster** element, the constraints apply when the port appears in a master interface. If the **modeConstraints** element is immediately followed by a **mirroredModeConstraints** element, the constraints defined in the **modeConstraints** element apply only when the port is used in a non-mirrored *mode* interface and the constraints defined in the **mirroredModeConstraints** element are used when the port is used in a mirrored mode interface. Otherwise, the constraints apply when the port appears in a *mode* interface or a mirrored-*mode* interface.

5.9 Transactional ports

5.9.1 Schema

The following schema defines the information contained in the **transactional** element, which may appear within a **port** within an abstraction definition (**abstractionDefinition/ports/port**).



5.9.2 Description

The **transactional** element defines a logical transactional port of the abstraction definition. This logical transactional port may provide optional constraints for a transactional port, to which it is mapped inside a component or abstractor's **busInterface**. The **transactional** element also contains the following elements and attributes:

- a) The **qualifier** (optional) element indicates which type of information this transactional port carries. It contains either or both of the following elements:
 - 1) **isAddress** (optional; type: *boolean*) specifies the port contains address information.
 - 2) **isData** (optional; type: *boolean*) specifies the port contains data information.
- b) **onSystem** (optional) defines constraints for this transactional port if it is present in a system bus interface with a matching group name.
 - 1) The **group** (type: *Name*) attribute indicates the group name for the transactional port. It distinguishes between different sets of system interfaces. Usually, all the arbiter ports are processed together, or all the clock or reset ports are processed together. So, this is really a mechanism to specify any sort of non-standard bus interface capabilities for the interconnect. The **group** name shall match the one specified in the bus definition **systemGroupName**.
 - 2) The group **transactionalPort** specifies what elements are used in this port. See [5.10](#).
- c) **onMaster** (optional) defines constraints for this transactional port when present in a master bus interface. The group **transactionalPort** specifies what elements are used in this port. See [5.10](#).
- d) **onSlave** (optional) defines constraints for this transactional port when present in a slave bus interface. The group **transactionalPort** specifies what elements are used in this port. See [5.10](#).

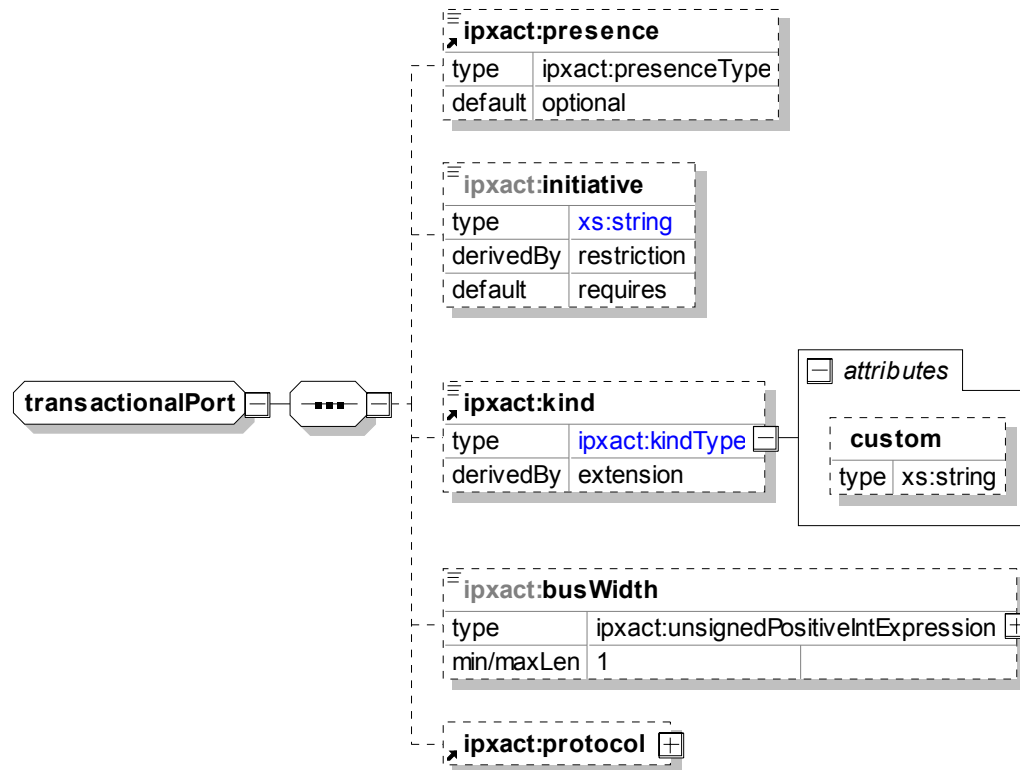
NOTE—The **onMaster**, **onSlave**, and **onSystem** elements associated with each logical port provide optional constraints. If any of these is missing, there are no constraints for how the port appears on interfaces with that mode (master, system, or slave). If no **onSystem** constraint is specified with a particular group, there are no constraints for system interfaces in that group. The abstraction definition author has the choice of how far to constrain the definitions. Generally speaking, more constraints in the definitions reduce implementation flexibility for whoever is creating IP with interface that conform to the abstraction definition.

See also [SCR 6.13](#), [SCR 6.14](#), and [SCR 6.16](#).

5.10 Transactional port group

5.10.1 Schema

The following schema defines the information contained in the **transactionalPort** group, which may appear within an **onMaster**, **onSlave**, or **onSystem** element within an abstraction definition (**abstractionDefinition/ports/port/transactional/onmode**).



5.10.2 Description

A **transactionalPort** group contains elements defining constraints associated with a transactional logical port within an **abstractionDefinition**. It contains the following elements:

- presence** (optional; default: **optional**) provides the capability to require or forbid a port to appear in a **busInterface**. Its three possible values are **illegal**, **required**, or **optional**. If this element is not present, its effective value is **optional**.
- initiative** (optional; type: **string**; default: **requires**) defines the type of access: **requires**, **provides**, or **both**. For example, a SystemC `sc_port` is defined using **requires**, since it requires a SystemC interface.
- kind** (optional) specifies the transactional port's type. It can take one of the following enum values: **tlm_port**, **tlm_socket**, **simple_socket**, **multi_socket**, or **custom**. When **custom**, a *name* can be specified in the custom attribute. Also, the **tlm_port** should be used only for TLM1 notation; the other enum values should be used for TLM2 sockets.
- busWidth** (optional; type: **unsignedPositiveIntExpression** (see [C.3.9](#))) specifies the data width in bits, e.g., 32 or 64.
- protocol** (optional) defines how the information is transported by the transactional port. (see [6.12.18](#)).

See also [SCR 6.2](#), [SCR 6.3](#), [SCR 6.4](#), [SCR 6.8](#), and [SCR 6.17](#).

5.11 Extending bus and abstraction definitions

5.11.1 Extending bus definitions

Bus definitions may use the **extends** element to create a family of compatible interconnectable bus definitions. A bus definition (B) extends another existing bus definition (A) by specifying the **extends** element in the B bus definition's element list. Bus definition B is referred to as the *extending* bus definition, and bus definition A is referred to as the *extended* bus definition. For two bus definitions related by the **extends** relation to be interconnectable, they need to be in a direct line of descent in the hierarchical *extension* tree, as illustrated in [Figure 8](#).

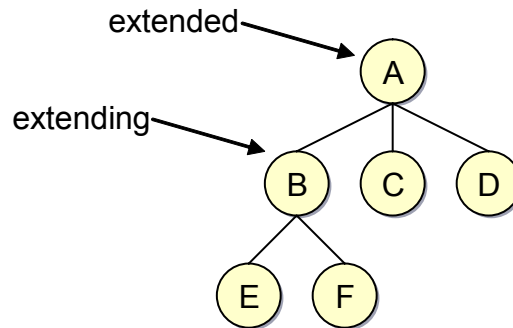


Figure 8—Extends relation hierarchy tree

In [Figure 8](#), bus definition B extends bus definition A. Bus interfaces of bus definition E shall be connected only with bus interfaces of bus definitions E, B, and A. By the same token, bus interfaces of bus definition F shall be connected only with bus interfaces of bus definitions F, B, and A.

5.11.2 Extending abstraction definitions

The **abstractionDefinition** that references the *extended* **busDefinition** via the **busType** element is referred to as the *extended* **abstractionDefinition**. The bus definition writer shall supply an **abstractionDefinition** that references the *extending* **busDefinition**, and it is referred to as the *extending* **abstractionDefinition**. The *extending* **abstractionDefinition** shall reference the *extended* **abstractionDefinition** via its **extends** element. An example of extending is shown in [Figure 9](#).¹⁵

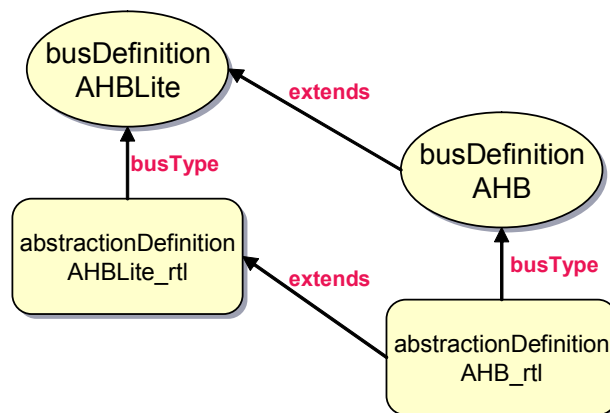


Figure 9—Example of extending

¹⁵AHB = AMBA® high-speed bus. AMBA is an open specification on-chip backbone for interconnecting IP blocks. AMBA is a registered trademark of ARM Limited. This information is given for the convenience of users of this standard and does not constitute an endorsement by the IEEE of this product. Equivalent products may be used if they can be shown to lead to the same results.

The *extending* bus definition and abstraction definition pair shall be able to stand on its own independent of the *extended* bus definition and abstraction definition pair; therefore, all the elements and attributes of the *extended* bus definition and abstraction definition pair shall be specified in the *extending* bus definition and abstraction definition pair. Also, all the ports in the *extended* abstraction definition shall be explicitly defined in the *extending* abstraction definition. Some of the elements and attributes of the *extending* bus definition and abstraction definition pair may be modified from the *extended* bus definition and abstraction definition pair, while others may not.

See also [SCR 3.16](#).

5.11.3 Modifying definitions

[Table 1](#) specifies which elements and attributes may be modified in a bus definition. The *Modifiable* column should be checked after elaboration.

Table 1—Elements of *extending* bus definition

Item	Modifiable	Note
directConnection	No	
broadcast	No	
isAddressable	No	
maxMasters	Yes	Smaller number applies when connecting interfaces of extended bus definitions.
maxSlaves	Yes	Smaller number applies when connecting interfaces of extended bus definitions.
systemGroupNames	Yes	New group names may be added; group names not specified are not allowed by this bus definition.
description	Yes	
parameters	Yes	
assertions	Yes	
vendorExtensions	Yes	

[Table 2](#) specifies which elements and attributes may be modified in an abstraction definition. The *Modifiable* column should be checked after elaboration.

Table 2—Elements of *extending* abstraction definition

Item	Modifiable	Note
ports	Yes	See Table 3 and SCR 6.11 .
description	Yes	
parameters	Yes	
assertions	Yes	
vendorExtensions	Yes	