

INTERNATIONAL IEEE Std 1685™-2009 STANDARD

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



THIS PUBLICATION IS COPYRIGHT PROTECTED
Copyright © 2009 IEEE

All rights reserved. IEEE is a registered trademark in the U.S. Patent & Trademark Office, owned by the Institute of Electrical and Electronics Engineers, Inc. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the IEC Central Office. Any questions about IEEE copyright should be addressed to the IEEE. Enquiries about obtaining additional rights to this publication and other information requests should be addressed to the IEC or your local IEC member National Committee.

IEC Central Office
3, rue de Varembe
CH-1211 Geneva 20
Switzerland
Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00
info@iec.ch
www.iec.ch

Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue
New York, NY 10016-5997
United States of America
stds.info@ieee.org
www.ieee.org

About the IEC

The International Electrotechnical Commission (IEC) is the leading global organization that prepares and publishes International Standards for all electrical, electronic and related technologies.

About IEC publications

The technical content of IEC publications is kept under constant review by the IEC. Please make sure that you have the latest edition, a corrigenda or an amendment might have been published.

IEC Catalogue - webstore.iec.ch/catalogue

The stand-alone application for consulting the entire bibliographical information on IEC International Standards, Technical Specifications, Technical Reports and other documents. Available for PC, Mac OS, Android Tablets and iPad.

IEC publications search - www.iec.ch/searchpub

The advanced search enables to find IEC publications by a variety of criteria (reference number, text, technical committee,...). It also gives information on projects, replaced and withdrawn publications.

IEC Just Published - webstore.iec.ch/justpublished

Stay up to date on all new IEC publications. Just Published details all new publications released. Available online and also once a month by email.

Electropedia - www.electropedia.org

The world's leading online dictionary of electronic and electrical terms containing more than 30 000 terms and definitions in English and French, with equivalent terms in 15 additional languages. Also known as the International Electrotechnical Vocabulary (IEV) online.

IEC Glossary - std.iec.ch/glossary

More than 60 000 electrotechnical terminology entries in English and French extracted from the Terms and Definitions clause of IEC publications issued since 2002. Some entries have been collected from earlier publications of IEC TC 37, 77, 86 and CISPR.

IEC Customer Service Centre - webstore.iec.ch/csc

If you wish to give us your feedback on this publication or need further assistance, please contact the Customer Service Centre: csc@iec.ch.

INTERNATIONAL IEEE Std 1685™-2009 STANDARD

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

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

ICS 25.040

ISBN 978-2-8322-2265-2

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

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.....	21
4.	Interoperability use model.....	23
4.1	Roles and responsibilities.....	23
4.2	IP-XACT IP exchange flows.....	24
5.	Interface definition descriptions.....	27
5.1	Definition descriptions.....	27
5.2	Bus definition.....	27
5.3	Abstraction definition.....	30
5.4	Ports.....	31
5.5	Wire ports.....	32
5.6	Qualifiers.....	34
5.7	Wire port group.....	36
5.8	Wire port mode constraints.....	38
5.9	Wire port mirrored-mode constraints.....	39
5.10	Transactional ports.....	41
5.11	Transactional port group.....	43
5.12	Extending bus and abstraction definitions.....	44
5.13	Clock and reset handling.....	47
6.	Component descriptions.....	49
6.1	Component.....	49
6.2	Interfaces.....	52
6.3	Interface interconnections.....	52
6.4	Complex interface interconnections.....	54
6.5	Bus interfaces.....	56
6.6	Component channels.....	67
6.7	Address spaces.....	69
6.8	Memory maps.....	81
6.9	Remapping.....	97
6.10	Registers.....	102
6.11	Models.....	120
6.12	Component generators.....	151
6.13	File sets.....	153
6.14	Choices.....	165

6.15 White box elements	167
6.16 White box element reference	168
6.17 CPUs	170
7. Design descriptions	171
7.1 Design	171
7.2 Design component instances	173
7.3 Design interconnections	175
7.4 Active, monitored, and monitor interfaces	176
7.5 Design ad hoc connections	178
7.6 Design hierarchical connections	180
8. Abstractor descriptions	183
8.1 Abstractor	183
8.2 Abstractor interfaces	185
8.3 Abstractor models	187
8.4 Abstractor views	189
8.5 Abstractor ports	191
8.6 Abstractor wire ports	193
8.7 Abstractor generators	195
9. Generator chain descriptions	199
9.1 generatorChain	199
9.2 generatorChainSelector	201
9.3 generatorChain component selector	202
9.4 generatorChain generator	203
10. Design configuration descriptions	207
10.1 Design configuration	207
10.2 designConfiguration	207
10.3 generatorChainConfiguration	209
10.4 interconnectionConfiguration	211
11. Addressing and data visibility	213
11.1 Calculating the bit address of a bit in a memory map	213
11.2 Calculating the bus address at the slave bus interface	214
11.3 Address modifications of an interconnection	214
11.4 Address modifications of a channel	215
11.5 Addressing in the master	216
11.6 Visibility of bits	216
11.7 Address translation in a bridge	218
Annex A (informative) Bibliography	219
Annex B (normative) Semantic consistency rules	221
Annex C (normative) Common elements and concepts	245
Annex D (normative) Types	263
Annex E (normative) Dependency XPATH	267

Annex F (informative) External bus with an internal/digital interface	271
Annex G (normative) Tight generator interface.....	273
Annex H (informative) Bridges and channels.....	351
Annex I (informative) IEEE List of Participants.....	361



IP-XACT, STANDARD STRUCTURE FOR PACKAGING, INTEGRATING, AND REUSING IP WITHIN TOOL FLOWS

FOREWORD

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

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

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

- 2) The formal decisions of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees. The formal decisions of IEEE on technical matters, once consensus within IEEE Societies and Standards Coordinating Committees has been reached, is determined by a balanced ballot of materially interested parties who indicate interest in reviewing the proposed standard. Final approval of the IEEE standards document is given by the IEEE Standards Association (IEEE-SA) Standards Board.
- 3) IEC/IEEE Publications have the form of recommendations for international use and are accepted by IEC National Committees/IEEE Societies in that sense. While all reasonable efforts are made to ensure that the technical content of IEC/IEEE Publications is accurate, IEC or IEEE cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
- 4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications (including IEC/IEEE Publications) transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC/IEEE Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
- 5) IEC and IEEE do not provide any attestation of conformity. Independent certification bodies provide conformity assessment services and, in some areas, access to IEC marks of conformity. IEC and IEEE are not responsible for any services carried out by independent certification bodies.
- 6) All users should ensure that they have the latest edition of this publication.
- 7) No liability shall attach to IEC or IEEE or their directors, employees, servants or agents including individual experts and members of technical committees and IEC National Committees, or volunteers of IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board, for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC/IEEE Publication or any other IEC or IEEE Publications.
- 8) Attention is drawn to the normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.
- 9) Attention is drawn to the possibility that implementation of this IEC/IEEE Publication may require use of material covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. IEC or IEEE shall not be held responsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity or scope of Patent Claims or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance, if any, or in any licensing agreements are reasonable or non-discriminatory. Users of this standard are expressly advised that determination of the validity of any patent rights, and the risk of infringement of such rights, is entirely their own responsibility.

International Standard IEC 62014-4/ IEEE Std 1685-2009 has been processed through IEC technical committee 91: Electronics assembly technology, under the IEC/IEEE Dual Logo Agreement.

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

IEEE Std	FDIS	Report on voting
1685 (2009)	91/1207/FDIS	91/1226/RVD

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

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

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

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

and the

IEEE Standards Association Corporate Advisory Group

Approved 9 December 2009

IEEE SA-Standards Board

Grateful acknowledgment is made to The SPIRIT Consortium, Inc., for permission to use the following source material:

IP-XACT 1.2 and IP-XACT 1.5

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, 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

AMBA is a registered trademark of ARM Limited.

Design Compiler and VCS are registered trademarks of Synopsys, Inc.

SystemC is a registered trademarks 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.

W3C is a registered trademark of the World Wide Web Consortium.

XMLSpy is a registered trademark of Altova GmbH in the U.S., the European Union and/or other countries.



IEEE Introduction

This introduction is not part of IEEE Std 1685-2009, 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 goal of this specification is to enable delivery of compatible IP descriptions from multiple IP vendors; better enable importing and exporting complex IP bundles to, from, and between EDA tools for system on chip (SoC) design environments (DEs); better express configurable IP by using IP meta-data; and better enable 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 is 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.

The SPIRIT Consortium has prepared a set of bus and abstraction definitions for several common buses. It is expected, over time, that those 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. The SPIRIT Consortium posts these for use by its members, with no warranty of suitability, but in the hope that these will be useful. The SPIRIT Consortium 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 SPIRIT Consortium Web site.^a

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

Notice to users

Laws and regulations

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

Copyrights

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

Updating of IEEE documents

Users of IEEE standards should be aware that these documents may be superseded at any time by the issuance of new editions or may be amended from time to time through the issuance of amendments, corrigenda, or errata. An official IEEE document at any point in time consists of the current edition of the document together with any amendments, corrigenda, or errata then in effect. In order to determine whether a given document is the current edition and whether it has been amended through the issuance of amendments, corrigenda, or errata, visit the IEEE Standards Association website at <http://ieeexplore.ieee.org/xpl/standards.jsp>, or contact the IEEE at the address listed previously.

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

Errata

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

Interpretations

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

Patents

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents or patent applications for which a license may be required to implement an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

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

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

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

1. Overview

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 and an *application programming interface* (API) to provide tool access to the meta-data. This schema provides a standard method to document IP that is compatible with automated integration techniques. The API provides a standard method 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 based on version 1.4 IP-XACT of The SPIRIT Consortium. The standard is independent of any specific design processes. It does not cover those 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) such 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.

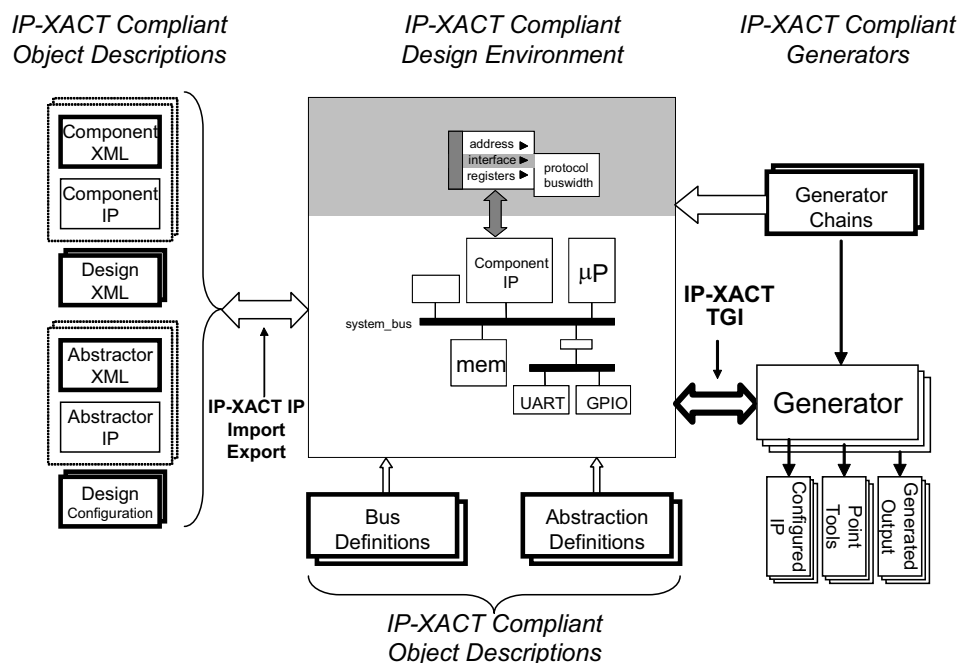


Figure 1—IP-XACT design environment

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

²IP-XACT uses the World Wide Web Consortium (W3C[®]) standard for the XML version 1.0 data (<http://www.w3.org/TR/2000/REC-xml-20001006>). 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.

provider and design intent by the IP user; and *design build*, which is the creation of a design (or design model) to those intentions.

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 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 v1.5 enabled, a system design tool shall support the import/export of IP expressed with valid IP-XACT v1.5 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 seven top-level schema definitions. Each schema definition can be used to create object descriptions of the corresponding type.

- A *bus definition* description defines the type attributes of an 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.

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.6](#) 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 → B) illustrate a reference of one object to another (e.g., reference of object B from object A).

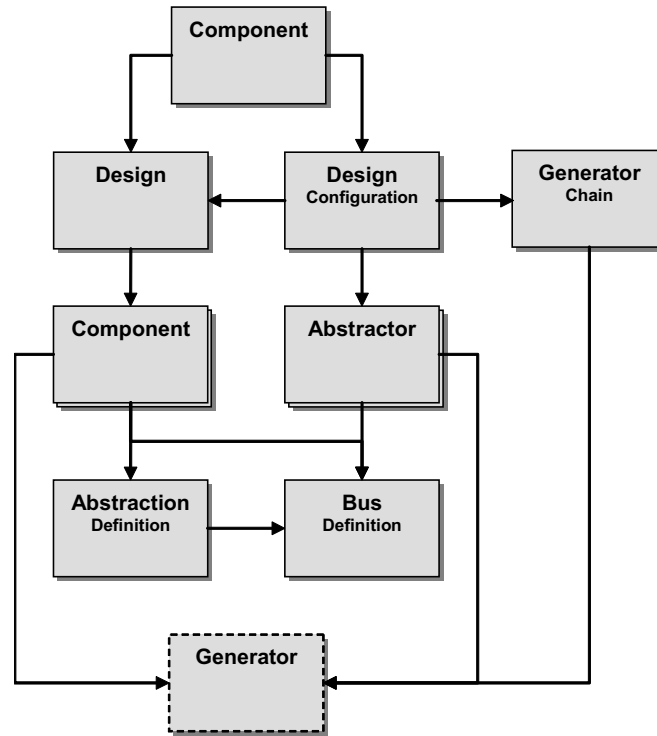


Figure 2—IP-XACT object interactions

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 netlister, 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 those 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 obvious 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.

1.3.6 Tight generator interface

The *tight generator interface* (TGI) is the method a generator uses to efficiently 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) standard version 1.2³ specified in the Web Services Description Language (WSDL) version 1.1.⁴ SOAP provides a simple means for sending XML-format messages using the Hypertext Transfer Protocol (HTTP) or other transport protocols. IP-XACT supports using an HTTP protocol 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 G](#).

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 ED community to refer to a reusable collection of design specifications that represent the behavior, properties, and/or representation 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 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 is a list of different views (levels of description and/or languages) to describe the IP object. In IP-XACT v1.5, these views include:
 - 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.

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.

³Available from the W3C Web site at <http://www.w3.org/TR/2007/REC-soap12-part1-20070427/>.

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

⁵SystemC is a registered trademarks 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.

1.4 IP-XACT Enabled implementations

Complying with the rules outlined in this subclause allows the provider 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 only be used 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 by the XSL Transform (XSLT) version 1.0,⁷ which is provided with the IP-XACT release to convert descriptions from one version to the next. 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:

- a) Shall follow the parse validity requirements shown in [1.4](#).
- b) Shall only create IP that is IP-XACT Enabled.
- c) When modifying any existing IP-XACT descriptions, shall do so without losing any preexisting information. In particular, it shall preserve any vendor extension data included in the existing IP-XACT description.
- d) Shall support the IP-XACT generator interfaces fully for interaction with underlying database.
- e) Shall be able to invoke all IP-XACT Enabled generators.

XPATH version 1.0⁸ support is required for DE-compliance.

⁷Available from the W3C Web site at <http://www.w3.org/TR/1999/REC-xslt-19991116>.

⁸Available from the W3C Web site at <http://www.w3.org/TR/1999/REC-xpath-19991116>.

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 IP-XACT Enabled DE (see [1.4.1](#)), an IP-XACT Enabled point tool:

- a) Shall follow the parse validity requirements shown in [1.4](#).
- b) Shall only create IP that is IP-XACT Enabled.
- c) When modifying any existing IP-XACT descriptions, shall do so 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:

- a) Shall have an IP-XACT description that follows the description and semantic validity requirements shown in [1.4](#).
- b) Shall only use IP-XACT Enabled generators for any generators associated with this IP.

XML descriptions compliant to 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:

- a) Shall only create IP that is IP-XACT Enabled.
- b) When modifying any existing IP-XACT descriptions, shall do so without losing any preexisting information. In particular, it shall preserve any vendor extension data included in the existing IP-XACT description.
- c) Shall be callable though the IP-XACT TGI (see [Annex G](#)).
- d) Shall only communicate with the DE that invoked it through the IP-XACT TGI (see [Annex G](#)).

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 document are to be interpreted as described in the IETF Best Practices Document 14, RFC-2119 [B5].⁹

1.5.3 Syntax examples

Any syntax examples shown in this standard are for information only and are only intended to illustrate the use of such syntax.

1.5.4 Graphics used to document the schema

The W3C Web site¹⁰ 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 scope and visibility issues, the information in the figures and the schema files is intended to be identical. Where the figures and schema are in conflict, the XML schema file shall take precedence.¹¹

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.

1.5.4.4 Namespace

Each element, attribute, type, or group has a name, which is preceded by a namespace and separated from the name by a colon (:). For the examples in 1.5.4.5, *xyz* is used as the namespace for all of the items

⁹The number in brackets correspond to those of the bibliography in Annex A.

¹⁰Available from the W3C Web site at <http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/>.

¹¹The 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.

whereas the standard uses **spirit**. Within the text of this standard, the namespace is not written when describing an item; it is only shown in examples.

1.5.4.5 Diagrams

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

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**.

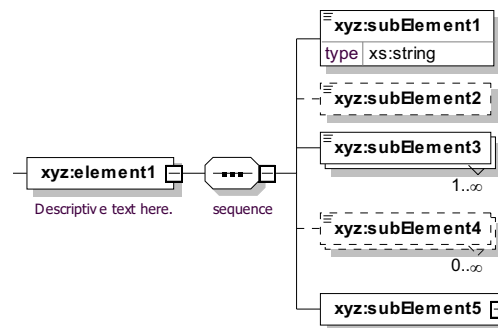


Figure 3—Sequence-compositor

- 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-act* 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.

[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.

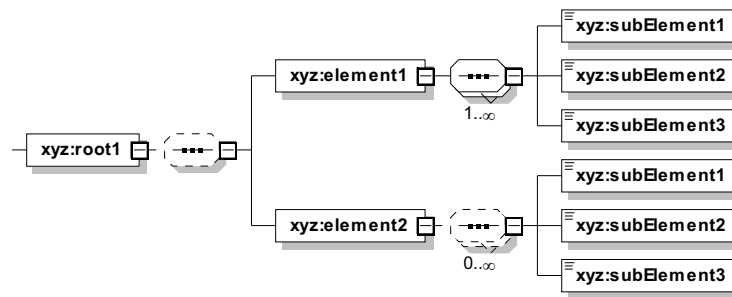


Figure 4—Sequence-compositor variations

- **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 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 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.

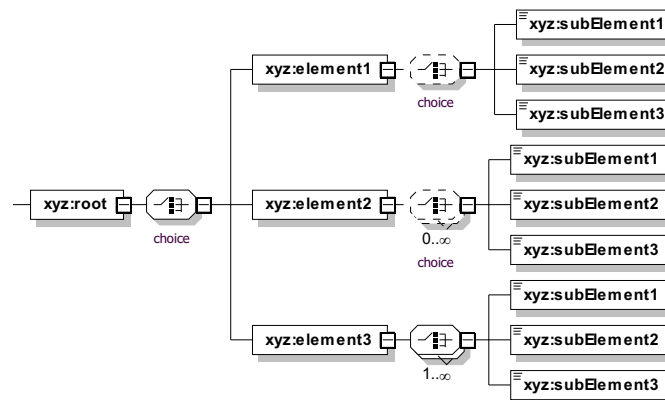


Figure 5—Choice-compositor variations

- 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. **attribute1** also has a type defined of *integer* and 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.

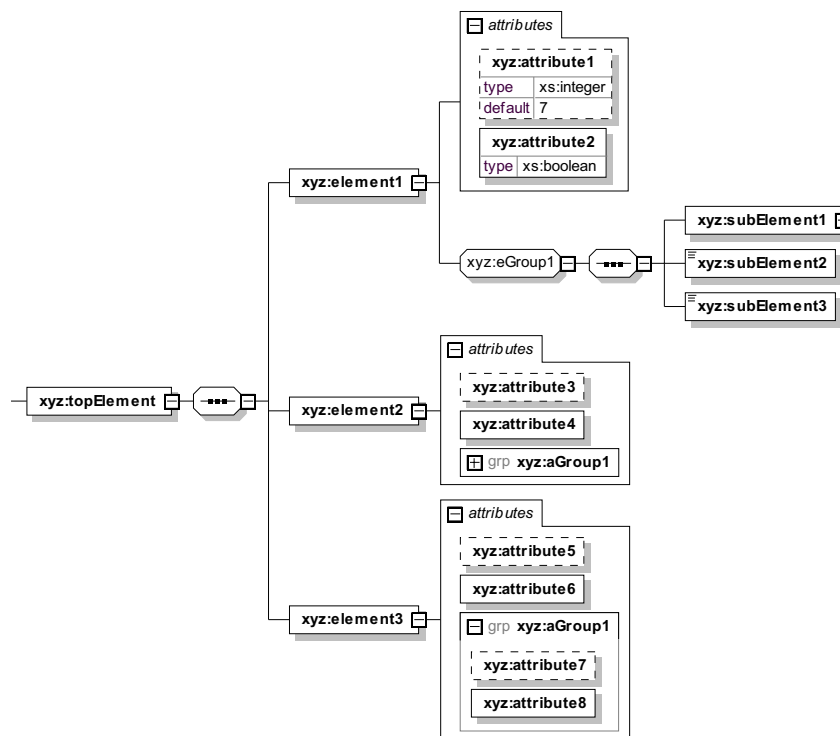


Figure 6—Attributes, groups, and attributeGroups

- 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.

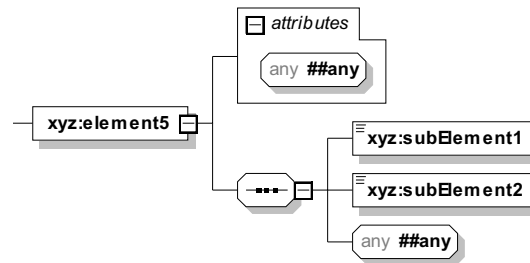


Figure 7—Wildcards

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).
- Syntactic keywords and tokens in the formal language definitions 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 addressing and data visibility.
- Annexes. Following [Clause 11](#) 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, so each referenced document is cited in text and its relationship to this document is explained). For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

Simple Object Access Protocol (SOAP) version 1.2 specification, available from the W3C Web site at <http://www.w3.org/TR/2007/REC-soap12-part1-20070427/>.

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

XML schema specification, available from the W3C Web site at <http://www.w3.org/TR/2004/REC-xmlschema-0-20041028>; <http://www.w3.org/TR/2004/REC-xmlschema-1-20041028>; <http://www.w3.org/TR/2004/PER-xmlschema-2-20040318>.

XML version 1.0 specification, available from the W3C Web site at <http://www.w3.org/TR/2000/REC-xml-20001006>.

XPATH version 1.0 specification, available from the W3C Web site at <http://www.w3.org/TR/1999/REC-xpath-19991116>.

XSLT version 1.0 specification, available from the W3C Web site at <http://www.w3.org/TR/1999/REC-xslt-19991116>.



3. Definitions, acronyms, and abbreviations

For the purposes of this document, the following terms and definitions apply. *The IEEE Standards Dictionary: Glossary of Terms & Definitions* should be referenced for terms not defined in this clause.¹²

3.1 Definitions

abstraction definition: An object that describes a representation of **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: Directly connects two **ports** without the use of **bus interfaces** or **interconnections**. Wire ad hoc connections have a wire protocol and transactional connections have a transactional connection.

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

architectural rules: Generic rules that define how **subsystems** relate to **platforms** that relate to **components** of system design.

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 only have direct interfaces. *Syn:* **bus bridge**.

bus: A collection of **ports** used to connect blocks connected to it involving both hardware and software protocols. Within IP-XACT, buses are **components**.

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 **IP** to a **bus**. **Components** are connected together by linking the bus interfaces together. There are three different classes of bus interfaces: master, slave, and system with two flavors: direct and mirrored.

channel: A special **object** that can be used to describe multi-point connections between regular components, which may require some interface adaptation. A channel connects **component master**, **slave**, and **system interfaces** on the same **bus**. A channel can also represent a simple wiring interconnection or a more complex structure, such as a bus. A channel can only have one address space. Channel interfaces are always mirrored interfaces. A channel supports memory mapping and remapping.

component: The central place holder for object **meta-data** and its bus and generator **interfaces**. Components are used to describe cores, peripherals, and buses. Components may reference designs to create hierarchy. *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.

¹²The *IEEE Standards Dictionary: Glossary of Terms & Definitions* is available at <http://shop.ieee.org/>.

configurable IP: IP that contains **configurable elements** and 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 manager: An object that creates and manages top-level meta-description of **system on chip** (SoC) design. It can annotate SoC schema with details of a specific SoC design including: IP versions, IP views, IP configuration, IP connectivity, and IP constraints. It manages the launching of **IP generators** and **tool plug-ins**, and any meta-data updates occurring as a consequence of a launch. It also handles the updating and retrieval of relevant **IP meta-data** from the IP repository.

connection: Generally describes a communication mechanism between one or more components.

constraint: Defines 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**, the **connections** between these components, and the **interfaces** exported by the system or subsystem.

design configuration: This description 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 and can specify a **view** for the **component** instances and **abstractors** for each **interconnection**, and 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): The coordination of a set of tools and **IP**, or expressions of that IP (e.g., models) so the system-design and implementation flows of a SoC reuse-centric development flow is efficiently enabled. This is managed by creating and maintaining a meta-data description of the **SoC**.

endianness: **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.

electronic design intellectual property (IP): Used in the electronic design community to refer to a reusable collection of design specifications that represent the behavior, properties, and/or representation 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 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); design and verification flow information and scripts. IP-XACT distinguishes between fixed IP and configurable IP.

electronic system level (ESL): A high level of design modeling typically done with, but not limited to, SystemC or SystemVerilog design languages.

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

NOTE—See ISO/IEC 8879 [B12].¹³

external components: **Components** that do not end up on the **SoC**, but are needed for total system verification.

fixed IP: **IP** that has no elements that are configured by the **DE** or set by industry de facto tools.

generator: Combines **component meta-data** with **architectural rules** to provide a consistent system description that uses a specified **tight generator interface (TGI)** to generate specific design views or **configurations** for the purposes of supporting a number of design styles. The generator may add/remove/replace components, add/remove/replace interconnections, add/remove/replace project settings, and add/remove/replace persistent data.

generator API: This **API** provides a common interface for algorithmic code in a **generator** or **tool plug-in** to the SOAP interface of the **TGI**.

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**.

generator TGI: This SOAP messaging interface connects the **generators** and **tool plug-ins** to the **design environment (DE)**, allowing the execution of these scripts and code-elements against the SoC meta-description. The **DE** enables the registration of new generators or plug-ins, exporting **SoC meta-data** and updating that data following generator or plug-in execution, and handling generator or plug-in error conditions that relate to the meta-data description.

hierarchical child bus interface: A bus interface **IC** of component **CC** is a hierarchical child of bus interface **IP** of component **CP** if and only if **CP** contains a hierarchical view, the design description of which contains a hierarchical connection with interface name **IP**, component ref **CC**, and interface ref **IC**. A hierarchical child bus interface may be a hierarchical bus interface itself.

hierarchical child component: A hierarchical child of a component **C** is any component referenced in a design of **C**.

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

hierarchical descendant bus interface: A bus interface **DC** is a hierarchical descendant of bus interface **AC** if and only if **DC** is a hierarchical child of **AC** or a hierarchical child of a hierarchical descendant of **AC**.

hierarchical descendent component: A hierarchical descendent of a component is any hierarchical child of that component or any hierarchical child of any hierarchical descendent of the component

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.

¹³Notes in text, tables, and figures are given for information only and do not contain requirements needed to implement the standard.

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

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

interconnection: Defines the point-to-point **connection** between two **bus interfaces**.

interface connection: Bus interfaces with **bus definitions** and **abstraction definitions** can be listed in the design as connected to another compatible interface on another component. The listing of the **interconnection** creates a **connection** to that **interface**.

IP generators: Tools that create specific **IP** based upon **SoC meta-data** details entered into the **configuration manager**. IP generators serve as **interfaces** to IP repository for placing and retrieval of IP and can annotate completion details (e.g., generated IP or failure of generation of IP) back into the configuration manager.

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

IP platform architect: Creator of platform-based architectures.

IP provider: Creator and supplier of **IP**.

IP repository: Database of IP.

leaf component: **Components** that do not contain other IP-XACT descriptions.

legacy IP: **IP** that has no specific IP-XACT **meta-data view**.

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 the design history, locality, object association, configuration options, constraints against, and integration requirements of an **object**.

meta IP: **Meta-data** description of an **object**.

mirror interface: Has the same (or similar) **ports** to its related direct **bus interface**, but the port directions are reversed. So, 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 neither a **master**, **slave**, nor **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: **components**, **designs**, **busDefinitions**, **abstractionDefinitions**, **abstractors**, **designConfigurations**, and **generatorChains**. To be able to be uniquely referenced, each object has an 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**.

Open SystemC Initiative (OSCI): An independent non-profit organization composed of a broad range of companies, universities and individuals dedicated to supporting and advancing SystemC as an open source standard for system-level design.

NOTE—See Transaction-Level Model of SystemC [\[B13\]](#).

phantom port: An initiative of a port that indicates this port does not have a true connection to the implementation, e.g., the port does not appear on the VHDL entity.

phase number: Defines the sequence in which **generators** should be fired.

platform: Architectural (sub)system framework.

platform consumer: User/group that builds a **SoC** based on a particular **platform**.

platform provider: User/group that develops and delivers **platforms** to **platform consumers**.

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

port: Specifies 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 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.

SoC platform: The top netlist containing all the instances and **connections** of the **design**.

style sheets: How documents are presented on screens and in print.

subsystem: A set of connected **components** that have dependencies on other **IP**.

system: A configured set of connected **components**.

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).

system on chip (SoC): Also refers to a general system that may not be implemented on a chip, such as a **transaction-level modeling (TLM)** design.

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

tool plug-ins: Tools that integrate **IP**, based upon **SoC meta-data** details, and prep **IP** for animation (e.g., simulation or emulation), optimization (e.g., synthesis), and **verification** (e.g., regression-suite generation). They can also annotate completion details (e.g., integrated SoC IP or failure of integration) back into the **configuration manager**.

transactional port: A **port** that has a service name (which can specify the data type of the port) and a port initiative. 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 **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: Proving the correctness of construction of a set of **components**.

Vendor Library Name Version (VLNV): Each IP-XACT **object** is assigned a unique identifier that is defined in the header of each XML file.

verification: Proving the behavior of a set of connected **components**.

verification IP (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.

white box interface (WBI): Internal points in the **IP** to be probed or driven by verification tools and/or test benches.

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.

XPATH: An expression language used by **XSLT** to access or refer to parts of an XML document.

XSLT: XSL Transform is a particular program written in the XSL language for performing a transformation (from one version to the next).

3.2 Acronyms and abbreviations

AHB	AMBA [®] high-speed bus ¹⁴
APB	AMBA peripheral bus
API	application programmers interface
AXI	Advanced eXtensible Interface
DE	design environment
EDA	electronic design automation
ESL	electronic system level
HDL	hardware description language
IP	electronic design intellectual property
LAU	least addressable unit (of memory)
OSCI	Open SystemC Initiative
PV	programmer's view
PVT	programmer's view with timing
RAM	random access memory
ROM	read only memory
RTL	register transfer level (design)
SCR	semantic consistency rule
SoC	system on chip
TGI	tight generator interface
TLI	task level interface
TLM	transaction-level modeling
VIP	verification IP
VLNV	Vendor Library Name Version
WBI	white box interface
XSLT	XSL Transform
XML	eXtensible Markup Language
3MD	three levels of meta-data

¹⁴AMBA is an open specification on-chip backbone for interconnecting **intellectual property** (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.



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 electronic design automation (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 v1.5 HDL and TLM system design.

4.1.1 Component IP provider

This is a person, group, or company creating IP components or subsystems for integration into a SoC design. These IPs can be hardware components (processors, memories, buses, etc.), 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. Those 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 his 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 into the following different types.

- *Fixed IP* is IP that is straightforward to describe and exchange as there are no configurable parameters. No generators need to be provided.
- *Parameterized IP* are those IP blocks that do not need IP specific generators, but have standard customizations (where *standard* is defined as industry de facto tool support), i.e., no generators need be provided for SoC design tools that support these parameterizations. An example of a parameterized IP is an AHB/APB bridge with configurable bus widths, done with VHDL generics or Verilog parameters.
- *Configurable IP* is IP created or modified as a direct result of running an IP-specific generator to build the IP to the user's specified configuration. This IP usually requires generators to be provided with it. An example of a configurable IP is an AHB bus fabric component that has a selectable number of masters and slaves, and automatic generation of decode functionality.

4.1.2 SoC design IP provider

This 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 (customers, partners, other EDA tools, etc.). 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

This 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

This 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, and 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: the IP vendor who supplies 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: new source files, parameters, documentation, etc.).

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 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.

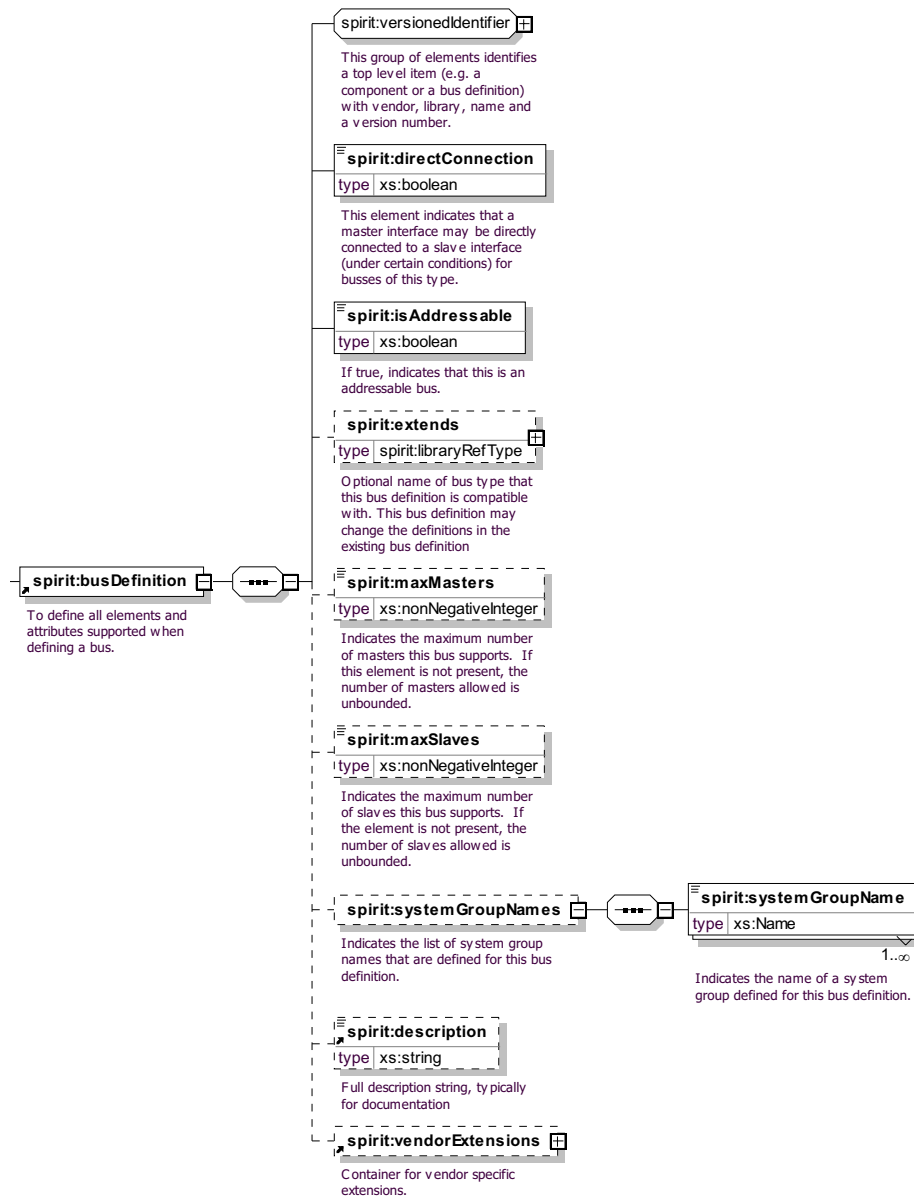
The *bus definition* description contains the high-level attributes of the interface, including items such as the connection method and indication of addressing.

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. 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 seven 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.6](#)
- directConnection** (mandatory) specifies what connections are allowed. The **directConnection** element is of type **boolean**. 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).
- isAddressable** (mandatory) specifies the bus has addressing information. The **isAddressable** element is of type **boolean** (see [6.3](#)). 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.

- d) **extends** (optional) specifies if this definition is an extension from another bus definition. The **extends** element is of type *libraryRefType* (see [C.7](#)), it contains four attributes to specify a unique VLNV. See also: [5.12](#).
- e) **maxMasters** specifies the maximum number of masters that are allowed on the bus. If the **maxMasters** element is not present, the numbers of masters is unbounded. The **maxMasters** elements is of type *nonNegativeInteger*.
- f) **maxSlaves** specifies the maximum number of slaves that are allowed to appear on the bus. If the **maxSlaves** element is not present, the numbers of slaves is unbounded. The **maxSlaves** elements is of type *nonNegativeInteger*.
- g) **systemGroupNames** (optional) defines an unbounded list of **systemGroupName** 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. The **systemGroupName** element is of type *Name*.
- h) **description** (optional) allows a textual description of the interface. The type of this element is *string*.
- i) **vendorExtensions** (optional) contains any extra vendor-specific data related to the interface. See [C.10](#).

See also: [SCR 1.3](#), [SCR 1.9](#), [SCR 1.11](#), and [SCR 6.17](#).

5.2.3 Example

This is an example of an AHB **busDefinition**.

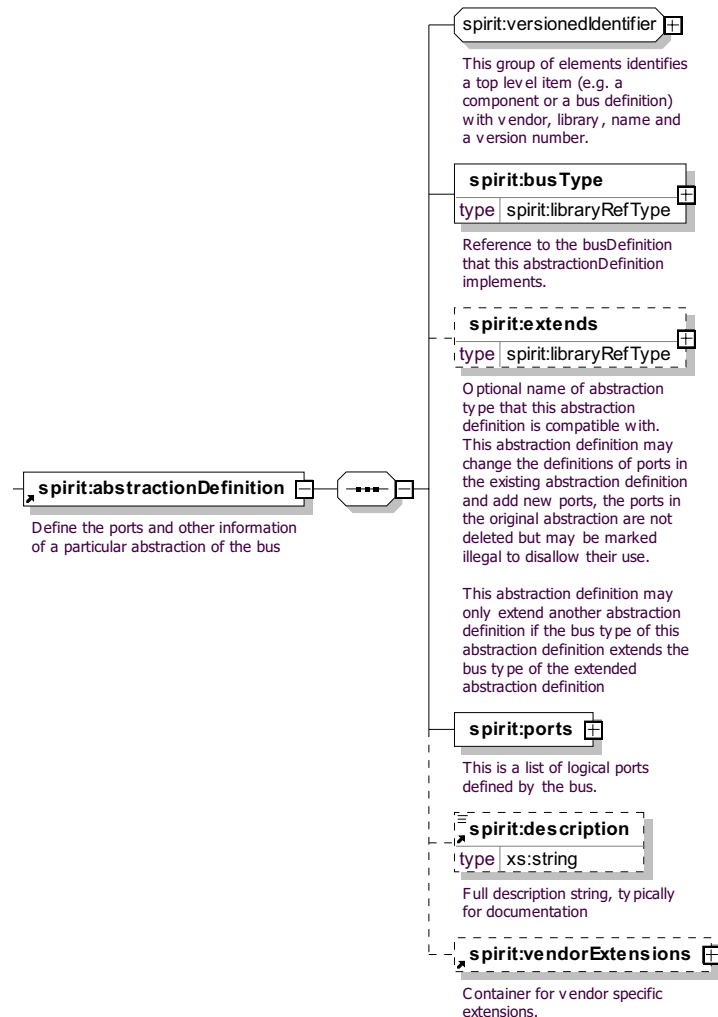
```
<?xml version="1.0" encoding="UTF-8" ?>
<spirit:busDefinition
  xmlns:spirit= http://www.spiritconsortium.org/XMLSchema/SPIRIT/1.5
  xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xsi:schemaLocation="http://www.spiritconsortium.org/XMLSchema/SPIRIT/1.5
  http://www.spiritconsortium.org/XMLSchema/SPIRIT/1.5/index.xsd">

  <spirit:vendor>amba.com</spirit:vendor>
  <spirit:library>AMBA</spirit:library>
  <spirit:name>AHB</spirit:name>
  <spirit:version>v1.0</spirit:version>
  <spirit:directConnection>false</spirit:directConnection>
  <spirit:isAddressable>true</spirit:isAddressable>
  <spirit:extends spirit:vendor="amba.com"
    spirit:library="AMBA"
    spirit:name="AHB-lite"
    spirit:version="v1.0" />
  <spirit:maxMasters>16</spirit:maxMasters>
  <spirit:maxSlaves>16</spirit:maxSlaves>
  <spirit:systemGroupNames>
    <spirit:systemGroupName>ahb_clk</spirit:systemGroupName>
    <spirit:systemGroupName>ahb_reset</spirit:systemGroupName>
  </spirit:systemGroupNames>
</spirit:busDefinition>
```

5.3 Abstraction definition

5.3.1 Schema

The following schema details the information contained in the **abstractionDefinition** element, which is one of the seven 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.6](#).
- busType** (mandatory) specifies the bus definition that this abstraction defines. The **busType** element is of type **libraryRefType** (see [C.7](#)); it contains four attributes to specify a unique VLVN. See also: [5.12](#).
- extends** (optional) specifies if this definition is an extension from another abstraction definition. The **extends** element is of type **libraryRefType** (see [C.7](#)), it contains four attributes to specify a unique

VLNV. 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: [5.12](#).

- d) **ports** (mandatory) is a list of logical ports, see [5.4](#).
- e) **description** (optional) allows a textual description of the interface. The type of this element is *string*.
- f) **vendorExtensions** (optional) contains any extra vendor-specific data related to the interface. See [C.10](#).

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 [5.10](#)). A *wire port* carries logic information or an array of logic information. A *transactional port* carries information that is represented on a higher level of abstraction.

See also: [SCR 1.9](#), [SCR 1.11](#), [SCR 1.13](#), [SCR 3.1](#), [SCR 3.16](#), [SCR 3.17](#), and [SCR 6.11](#).

5.3.3 Example

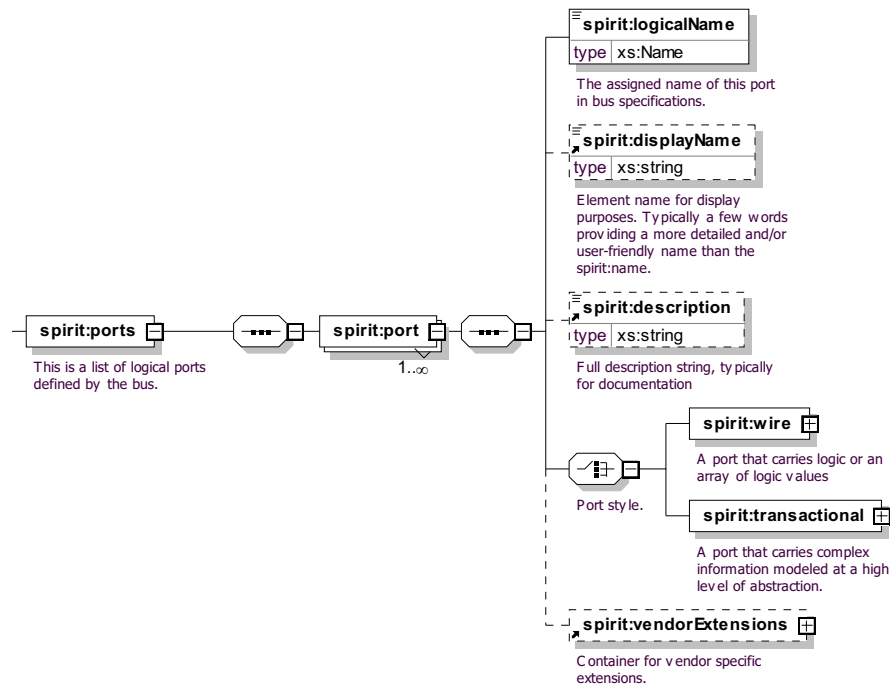
The following example shows an abstraction definition for the interrupt bus in the Leon2 TLM example.

```
<spirit:vendor>spiritconsortium.org</spirit:vendor>
<spirit:library>Leon</spirit:library>
<spirit:name>INT_PV</spirit:name>
<spirit:version>1.5</spirit:version>
<spirit:busType spirit:vendor="spiritconsortium.org"
spirit:library="Leon" spirit:name="Int" spirit:version="v1.0"/>
<spirit:ports>
  <spirit:port>
    <spirit:logicalName>INT_TRANSACTION</spirit:logicalName>
    <spirit:wire>
      <spirit:onMaster>
        <spirit:presence>required</spirit:presence>
        <spirit:direction>out</spirit:direction>
      </spirit:onMaster>
      <spirit:onSlave>
        <spirit:presence>required</spirit:presence>
        <spirit:direction>in</spirit:direction>
      </spirit:onSlave>
    </spirit:wire>
  </spirit:port>
</spirit:ports>
```

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

The **ports** element is an unbounded list of **port** elements. Each **port** element defines the logical port information for the containing abstraction definition. It contains the following elements.

- logicalName** (mandatory) 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**. The type of this element is *Name*.
- displayName** (optional) allows a short descriptive text to be associated with the port. The type of this element is *string*.
- description** (optional) allows a textual description of the port. The type of this element is *string*.
- Each **port** also requires a **wire** element or a **transactional** element to further describe the details about this port. See 5.5 or 5.10, 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.
- vendorExtensions** (optional) contains any extra vendor-specific data related to the port. See C.10.

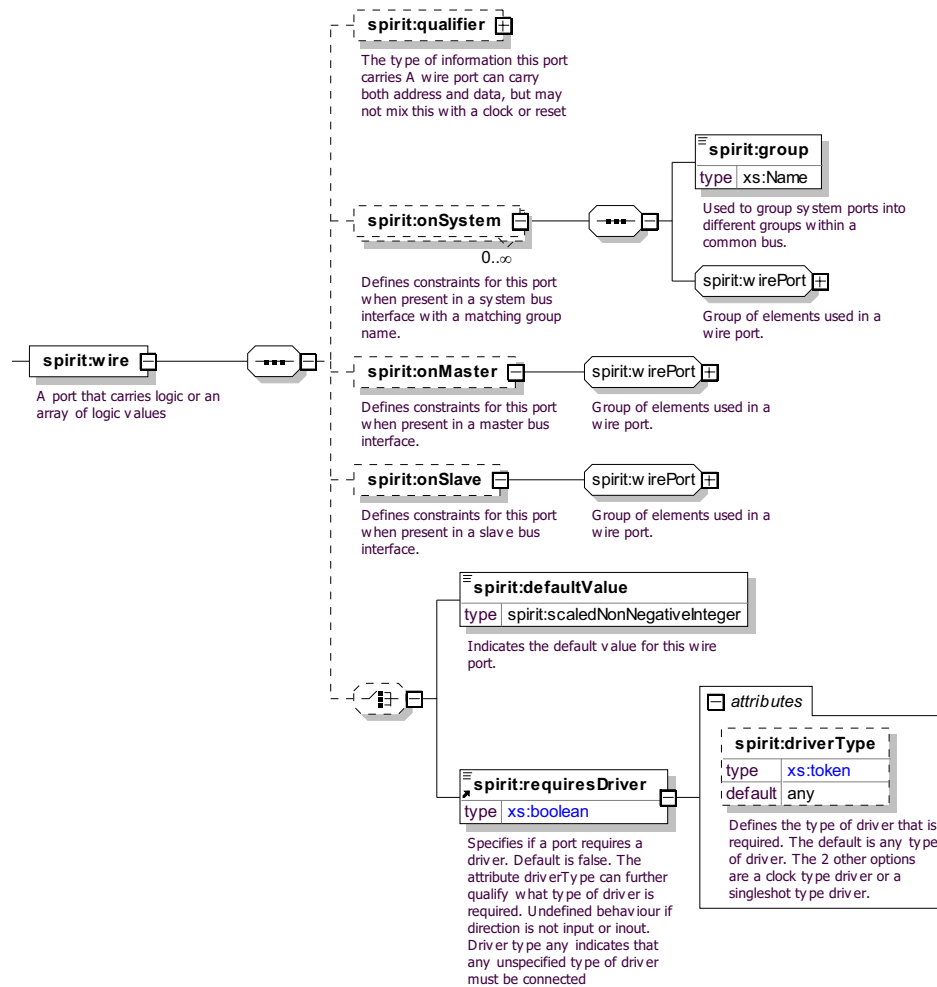
5.4.3 Example

See 5.3.3 for an example.

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) Either of the following two elements are allowed, but not both.
- 1) **defaultValue** (optional) 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 not an ad hoc connection to the corresponding physical port. The type of this element is *scaledNonNegativeInteger*.
 - 2) **requiresDriver** (optional) specifies whether the port requires a driver when used in a completed design. The type of this element is *boolean*. 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** (meaning any logic signal or value), **clock** (meaning a repeating type waveform), or **singleshot** (a non-repeating type waveform). If this element 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 are 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 6.15](#).

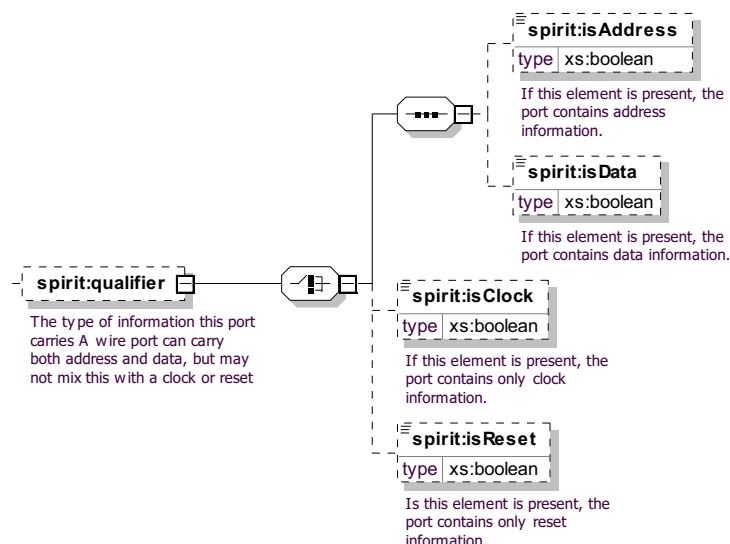
5.5.3 Example

See [5.3.3](#) for an example.

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), when **true**, specifies the port contains address information. This **qualifier** may be paired with the **isData** element (e.g., used with serial protocols). The type of this element is *boolean*. See also: [Clause 11](#).
- b) **isData** (optional), 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). The type of this element is *boolean*. See also: [Clause 11](#).
- c) **isClock** (optional), 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 only be applied to pure clock ports. This **qualifier** shall not be combined with other qualifiers. The type of this element is *boolean*.
- d) **isReset** (optional), 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 only be applied to pure reset ports. This **qualifier** shall not be combined with other qualifiers. The type of this element is *boolean*.

See also: [SCR 6.17](#), [SCR 9.1](#), [SCR 9.2](#), and [SCR 12.8](#).

5.6.3 Example

```
<spirit:port>
  <spirit:logicalName>Clock</spirit:logicalName>
  <spirit:wire>
    <spirit:qualifier>
      <spirit:isClock>true</spirit:isClock>
    </spirit:qualifier>
    <spirit:onSystem>
      <spirit:group>clk</spirit:group>
      <spirit:width>1</spirit:width>
      <spirit:direction>out</spirit:direction>
    </spirit:onSystem>
    <spirit:onMaster>
      <spirit:direction>in</spirit:direction>
    </spirit:onMaster>
    <spirit:onSlave>
      <spirit:direction>in</spirit:direction>
    </spirit:onSlave>
  </spirit:wire>
</spirit:port>
<spirit:port>
  <spirit:logicalName>Resetn</spirit:logicalName>
  <spirit:wire>
    <spirit:qualifier>
      <spirit:isReset>true</spirit:isReset>
    </spirit:qualifier>
    <spirit:onSystem>
      <spirit:group>reset</spirit:group>
      <spirit:width>1</spirit:width>
      <spirit:direction>out</spirit:direction>
```

```

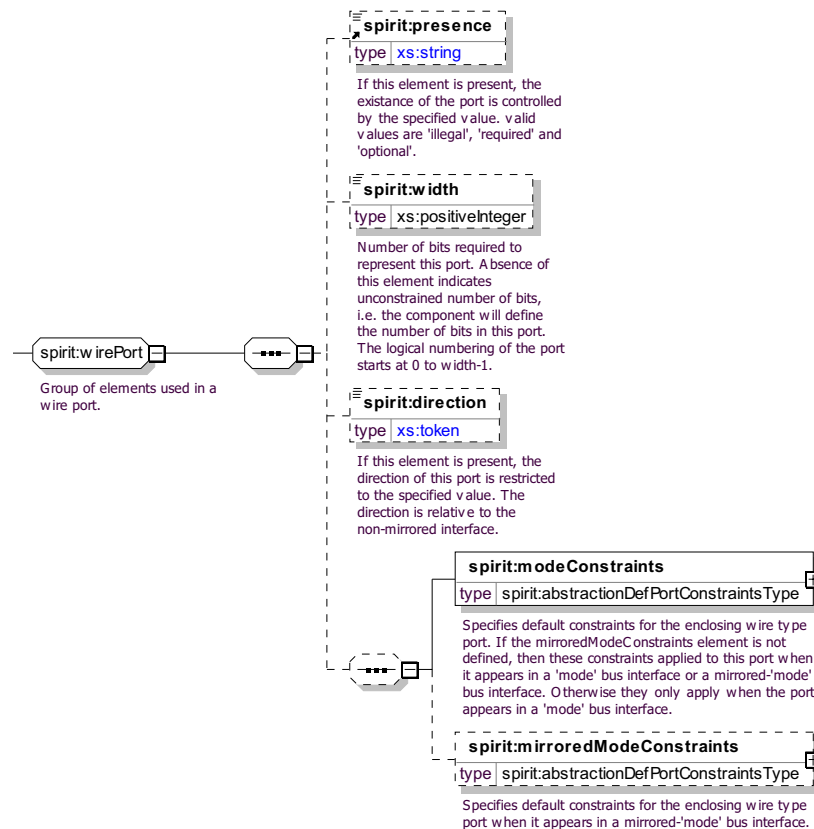
        </spirit:onSystem>
        <spirit:onMaster>
            <spirit:direction>in</spirit:direction>
        </spirit:onMaster>
        <spirit:onSlave>
            <spirit:direction>in</spirit:direction>
        </spirit:onSlave>
    </spirit:wire>
</spirit:port>
<spirit:port>
    <spirit:logicalName>Address</spirit:logicalName>
    <spirit:wire>
        <spirit:qualifier>
            <spirit:isAddress>true</spirit:isAddress>
        </spirit:qualifier>
        <spirit:onMaster>
            <spirit:direction>out</spirit:direction>
        </spirit:onMaster>
        <spirit:onSlave>
            <spirit:direction>in</spirit:direction>
        </spirit:onSlave>
    </spirit:wire>
</spirit:port>

```

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) 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) 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 $\text{width}-1$. 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. The **width** element is of type *positiveInteger*.
- direction** (optional) 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 constraints of this interface during synthesis. 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 5.9).

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 6.18](#).