IEEE P1685™/D5 Draft 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

Copyright © 2014 by the Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA
All rights reserved.

All rights reserved.This document is an unapproved draft of a proposed IEEE Standard. As such, this document is subject to change. USE AT YOUR OWN RISK! Because this is an unapproved draft, this document must not be utilized for any conformance/compliance purposes. Permission is hereby granted for IEEE Standards Committee participants to reproduce this document for purposes of IEEE standardization activities only. Prior to submitting this document to another standards development organization for standardization activities, permission must first be obtained from the Manager, Standards Licensing and Contracts, IEEE Standards Activities Department. Other entities seeking permission to reproduce this document, in whole or in part, must obtain permission from the Manager, Standards Licensing and Contracts, IEEE Standards Activities Department.

IEEE Standards Department
Copyright and Permissions
445 Hoes Lane, P.O. Box 1331
Piscataway, NJ 08855-1331, USA

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

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

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

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

Copyright © 2014 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. 1735-201y Published xx month 201y. Printed in the United States of America.

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

1685-201y PDF: ISBN TBD STD TBD
1685-201y Print: ISBN TBD STD TBD

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

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

Use of an IEEE Standard is wholly voluntary. The IEEE disclaims liability for any personal injury, property or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon this, or any other IEEE Standard document.

The IEEE does not warrant or represent the accuracy or content of the material contained herein, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained herein is free from patent infringement. IEEE Standards documents are supplied **"AS IS."**

The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

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

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration. A statement, written or oral, that is not processed in accordance with the IEEE-SA Standards Board Operations Manual shall not be considered the official position of IEEE or any of its committees and shall not be considered to be, nor be relied upon as, a formal interpretation of the IEEE. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that his or her views should be considered the personal views of that individual rather than the formal position, explanation, or interpretation of the IEEE.

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

Secretary, IEEE-SA Standards Board

445 Hoes Lane

Piscataway, NJ 08854

USA

Authorization to photocopy portions of any individual standard for internal or personal use is granted by The Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Introduction

This introduction is not part of IEEE Std P1685-2014, IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows.

The purpose of this standard is to provide the electronic design automation (EDA), semiconductor, electronic design intellectual property (IP) provider, and system design communities with a well-defined and unified specification for the meta-data that represents the components and designs within an electronic system. The 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, which originally developed the IP-XACT standard before merging with Accellera, was a consortium of electronic system, IP provider, semiconductor, and EDA companies. IP-XACT enables a productivity boost in design, transfer, validation, documentation, and use of electronic IP and covers components, designs, interfaces, and details thereof. The data specified by IP-XACT is extensible in locations specified in the schema.

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

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

Accellera has prepared a set of bus and abstraction definitions for several common buses. It is expected, over time, that 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. Accellera posts these for use by its members, with no warranty of suitability, but in the hope that these will be useful. Accellera will, from time-to-time, update these files and if a Standards body wishes to take over the work of definition, will transfer that work to that body.

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

[a]Available at http://www.accellera.org.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Notice to users

Laws and regulations

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

Copyrights

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

Updating of IEEE documents

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

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

Errata

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

Interpretations

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

Patents

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. A patent holder or patent applicant has filed a statement of assurance that it will grant licenses under these rights without compensation or under reasonable rates, with reasonable terms and conditions that are demonstrably free of any unfair discrimination to applicants desiring to obtain such licenses. Other Essential Patent Claims may exist for which a statement of assurance

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

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Participants

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

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

David Courtright
Joe Daniels
Edwin Dankert
Jean-Michel Fernandez
Jeffery Griffiths
Stephane Guntz
Prashant Karandikar
Erwin de Kock
Mark Noll
Kamlesh Pathak
Richard Weber

The rest of these lists will be set by IEEE prior to publication

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

TBD; show all voting entities.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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

chair, *Chair*
vice chair, *Vice Chair*
past chair, *Past Chair*
secretary, *Secretary*

IEEE to supply

*Member Emeritus

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

rep, *NRC Representative*
rep, *DOE Representative*
rep, *NIST Representative*

program manager
IEEE Standards Program Manager, Document Development

program manager
IEEE Standards Program Manager, Technical Program Development

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Contents

1. Overview ... 1
1.1 Scope ... 1
1.2 Purpose ... 2
1.3 Design environment ... 2
1.4 IP-XACT Enabled implementations ... 6
1.5 Conventions used ... 7
1.6 Use of color in this standard ... 12
1.7 Contents of this standard ... 12

2. Normative references ... 13

3. Definitions, acronyms, and abbreviations ... 15
3.1 Definitions ... 15
3.2 Acronyms and abbreviations ... 20

4. Interoperability use model ... 21
4.1 Roles and responsibilities ... 21
4.2 IP-XACT IP exchange flows ... 22

5. Interface definition descriptions ... 25
5.1 Definition descriptions ... 25
5.2 Bus definition ... 25
5.3 Abstraction definition ... 27
5.4 Ports ... 29
5.5 Wire ports ... 30
5.6 Qualifiers ... 31
5.7 Wire port group ... 32
5.8 Wire port mode (and mirrored mode) constraints ... 33
5.9 Transactional ports ... 34
5.10 Transactional port group ... 35
5.11 Extending bus and abstraction definitions ... 37
5.12 Clock and reset handling ... 40

6. Component descriptions ... 41
6.1 Component ... 41
6.2 Interfaces ... 44
6.3 Interface interconnections ... 44
6.4 Complex interface interconnections ... 46
6.5 Bus interfaces ... 48
6.6 Indirect interfaces ... 57
6.7 Component channels ... 59
6.8 Address spaces ... 60
6.9 Memory maps ... 67
6.10 Remapping ... 79
6.11 Registers ... 82
6.12 Models ... 96
6.13 Component generators ... 124
6.14 Choices ... 125
6.15 File sets ... 126
6.16 White box elements ... 134
6.17 White box element reference ... 135

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

6.18 CPUs . . . 136
6.19 Reset types . . . 137

7. Design descriptions . . . 139
7.1 Design . . . 139
7.2 Design component instances . . . 140
7.3 Design interconnections . . . 141
7.4 Active, hierarchical, monitored, and monitor interfaces . . . 143
7.5 Design ad hoc connections . . . 146
7.6 Port references . . . 148

8. Abstractor descriptions . . . 151
8.1 Abstractor . . . 151
8.2 Abstractor interfaces . . . 153
8.3 Abstractor models . . . 153
8.4 Abstractor views . . . 154
8.5 Abstractor ports . . . 155
8.6 Abstractor wire ports . . . 156
8.7 Abstractor generators . . . 158

9. Generator chain descriptions . . . 161
9.1 generatorChain . . . 161
9.2 generatorChainSelector . . . 163
9.3 generatorChain component selector . . . 164
9.4 generatorChain generator . . . 165

10. Design configuration descriptions . . . 167
10.1 Design configuration . . . 167
10.2 designConfiguration . . . 167
10.3 interconnectionConfiguration . . . 169
10.4 abstractorInstance . . . 170
10.5 viewConfiguration . . . 171

11. Catalog descriptions . . . 173
11.1 catalog . . . 173
11.2 ipxactFile . . . 174

12. Addressing . . . 177
12.1 Calculating the bit address of a bit in a memory map . . . 177
12.2 Calculating the bus address at the slave bus interface . . . 179
12.3 Calculating the address at the indirect interface . . . 179
12.4 Address modifications of a channel . . . 180
12.5 Addressing in the master . . . 180
12.6 Address translation in a bridge . . . 180

13. Data visibility . . . 181
13.1 Mapped address bits mask . . . 181
13.2 Address modifications of an interconnection . . . 182
13.3 Bit steering in a channel . . . 182
13.4 Visibility of bits . . . 182

Annex A(informative)Bibliography . . . 185

Annex B(normative) Semantic consistency rules . . . 187

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Annex C (normative) Common elements and concepts 217

Annex D (normative) Types 251

Annex E (normative) SystemVerilog expressions 253

Annex F (normative) Tight generator interface 267

Annex G (informative) External bus with an internal/digital interface 457

Annex H (informative) Bridges and channels 459

Annex I (informative) Examples 469

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

IEEE Draft Standard for 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) schema[1] for meta-data documenting *intellectual property* (IP) used in the development, implementation, and verification of electronic systems. This schema provides both a standard method to document IP that is compatible with automated integration techniques and a standard method (generators) for linking tools into a *system development* framework, enabling a more flexible, optimized development environment. Tools compliant with this standard will be able to interpret, configure, integrate, and manipulate IP blocks that comply with the IP meta-data description. The standard is independent of any specific design processes. It does not cover those behavioral characteristics of the IP that are not relevant to integration.

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

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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.

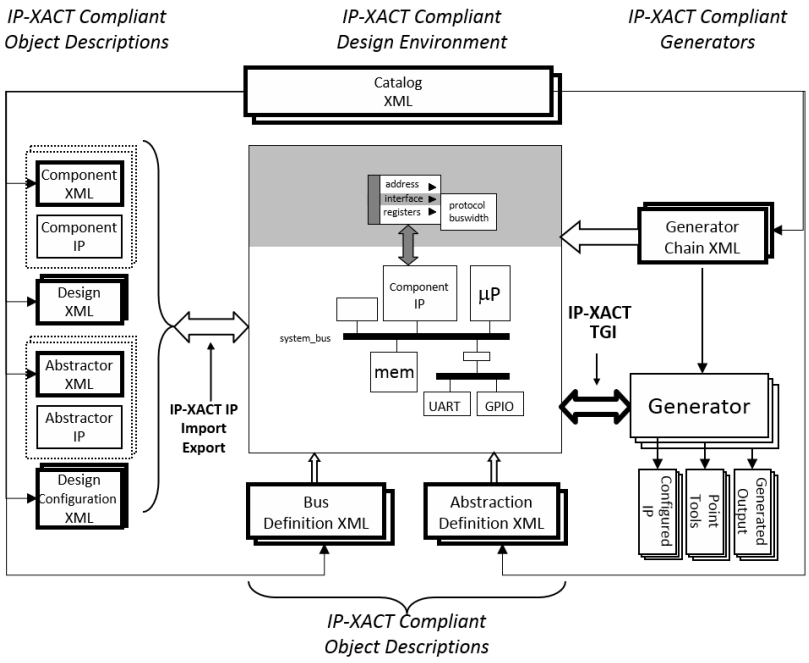


Figure 1—IP-XACT design environment

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

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

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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 enabled, a system design tool shall support the import/export of IP expressed with valid IP-XACT meta-data for both component IP and designs, and it needs to support the tight generator interface (TGI) for interfacing with external generators (to the DE).

1.3.2 IP-XACT object descriptions

The IP-XACT schema is the core of the IP-XACT specification. There are 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.
- A *catalog* description provides a mapping between IP-XACT VLNVs (see 1.3.3) and the physical location of the IP-XACT file defining the IP-XACT object with the given VLNV.

1.3.3 Object interactions

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

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

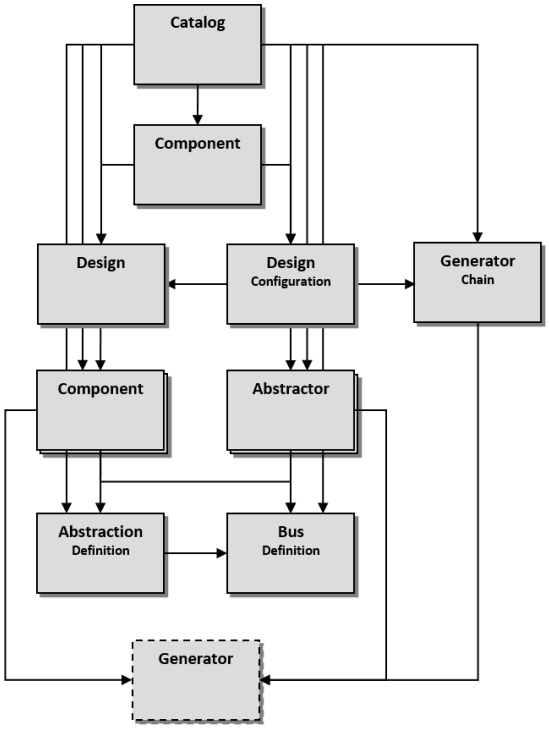


Figure 2—IP-XACT object interactions

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

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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

1.3.6 Tight generator interface

The *tight generator interface* (TGI) is the method a generator uses to 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)[3] specified in the Web Services Description Language (WSDL).[4] SOAP provides a simple means for sending XML-format messages using the Hypertext Transfer Protocol (HTTP) or other transport protocols. IP-XACT supports using 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 F.

1.3.7 Design intellectual property

IP-XACT is structured around the concept of IP reuse. *Electronic design intellectual property*, or IP, is a term used in the 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 SystemVerilog [5]
   2) Fixed HDL descriptions: Verilog®, VHDL[6]
   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, 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.

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

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

[5]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. 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.

[6]See Footnote 5.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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

1.4 IP-XACT Enabled implementations

Complying with the rules outlined in this subclause allows 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 to the highest schema version supported by the given tool. This is required for semantic consistency. Schema version translation can be done in a number of different ways, but the most common is to leverage the XSL Transformations (XSLT)[7] provided with the IP-XACT schema image. In addition, a DE or point tool may preserve information in the initial description for use outside of the scope of the IP-XACT specification.

1.4.1 Design environments

An IP-XACT Enabled DE:

a) Shall follow the parse validity requirements shown in 1.4.
b) Shall 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 be able to invoke IP-XACT Enabled generators with **apiType** of **none** (see 6.13.2).

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

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

An IP-XACT TGI Enabled DE:

e) Shall be considered as enabled for the base TGI if it supports all generators utilizing the `TGI_2014_BASE` **apiType** (see 6.13.2).

f) Shall be considered as enabled for the extended TGI if it supports all generators utilizing the `TGI_2014_BASE` or `TGI_2014_EXTENDED` **apiType**s (see 6.13.2).

1.4.2 Point tools

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

a) Shall follow the parse validity requirements shown in 1.4.

b) Shall 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 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 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 only communicate with the DE that invoked it through the IP-XACT TGI (see Annex F).

1.5 Conventions used

The conventions used throughout the document are included here.

IP-XACT is case-sensitive.

1.5.1 Visual cues (meta-syntax)

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

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

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

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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].[8]

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 (see also Annex I).

1.5.4 Graphics used to document the schema

The W3C Web site[9] specifies the XML schema language used to define the IP-XACT XML schemas. Normative details for compliance to the IP-XACT standard are contained in the schema files. Within this document, pictorial representations of the information in the schema files *illustrate* the structure of the schema and *define* any constraints of the standard. With the exception of visibility issues, the information in the figures and the schema files is intended to be identical (for a fully annotated version of the schema files, see [B9]). Where the figures and schema are in conflict, the XML schema file shall take precedence.[10]

1.5.4.1 Elements and attributes

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

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

1.5.4.2 Types

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

1.5.4.3 Groups

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

1.5.4.4 Namespace

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

[8]The number in brackets correspond to those of the bibliography in Annex A.

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

[10]The 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.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

items whereas the standard uses **ipxact**. Within the text of this standard, the namespace-prefix 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 of elements and attributes.

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

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**.

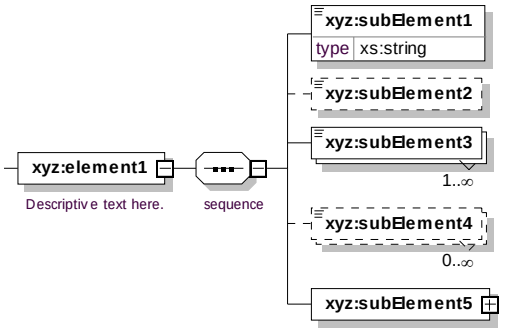


Figure 3—Sequence-compositor

a) **subElement1** is a mandatory element, as indicated by the solid line of the containing box. The type of the data contained in this element is set to ***string*** and it has a default value of `ip-xact` if the element is present, but left empty.

b) **subElement2** is an optional element, as indicated by the dashed-line of the containing box.

c) **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.

d) **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.

e) **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.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

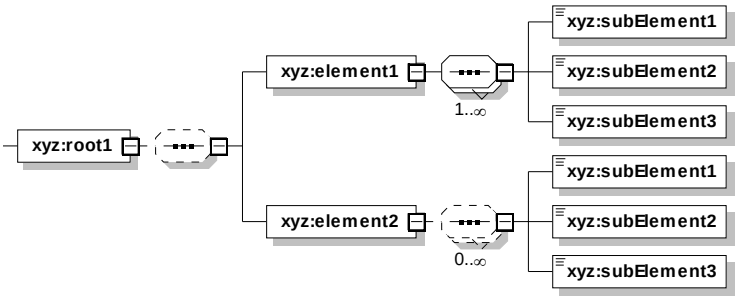


Figure 4—Sequence-compositor variations

— **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 then `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.

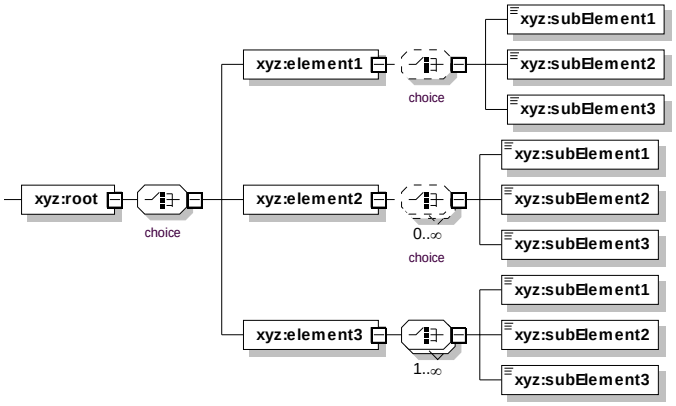


Figure 5—Choice-compositor variations

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

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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.

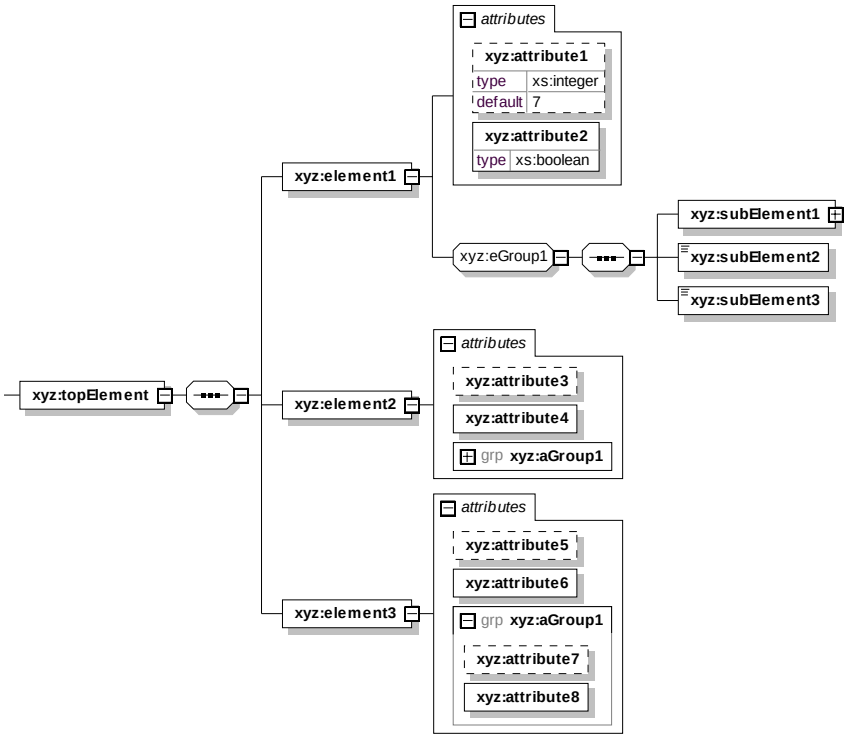


Figure 6—Attributes, groups, and attributeGroups

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 it 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.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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.

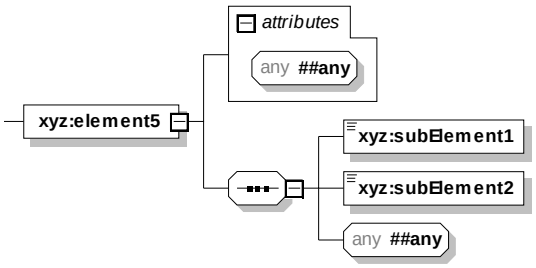


Figure 7—Wildcards

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 the catalog feature.
- Clause 12 defines addressing.
- Clause 13 defines data visibility.
- Annexes. Following Clause 13 are a series of annexes.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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.

IEEE Std 1800™, Standard for SystemVerilog-Unified Hardware Design, Specification, and Verification Language.[11, 12]

SOAP version 1.2 specification, available from the W3C Web site at http://www.w3.org/TR/soap12-part1/.

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

XML schema specification, available from the W3C Web site at http://www.w3.org/TR/xmlschema-0/; http://www.w3.org/TR/xmlschema-1/; http://www.w3.org/TR/xmlschema-2/.

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

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

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

XSLT version 1.0 specification, available from the W3C Web site at http://www.w3.org/TR/xslt.

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

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

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

3. Definitions, acronyms, and abbreviations

For the purposes of this document, the following terms and definitions apply. The IEEE Standards Dictionary Online should be consulted for terms not defined in this clause.[13]

3.1 Definitions

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

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 or more **port**s without the use of **bus interface**s or **interconnection**s.

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

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

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

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

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

bus interface: The **interface** of an **IP** to an **interconnection**. **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.

catalog: An object that defines a mapping from IP-XACT VLNV to the physical file in which the IP-XACT **object** is defined.

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

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

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

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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

configurable IP: **IP** that contains **configurable element**s, and/or an IP-specific generator capable of creating new components from the configured component and updating the design with the new version of the **component**. *Syn*: **configurable component**.

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

NOTE—See C.10.[14]

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 **component**s, and the **connection**s between these components and the **interface**s 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 **abstractor**s for each **interconnection**, and configure generator **chain**s. *Syn*: **configuration**.

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

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

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.

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

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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

NOTE—See ISO/IEC 8879 [B13].

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

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

generator chain: A hierarchical list of generators used to define the order for executing **generator**s. 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**.

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

NOTE—See C.1.

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

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

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

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

indirectly accessible memory map: In some cases, the contents of a **memory map** cannot be accessed at a fixed address via a **bus interface**. Instead the contents are indirectly accessible via indirect address and data fields. A **bus interface** accesses an indirectly accessible memory map via reads and writes to indirect address and data fields.

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

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

IP integrator: A party in the design process who receives 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.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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

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

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

meta-data: A tool-interpretable way of describing the design history, locality, object association, configuration options, constraints against, and integration requirements 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: **Bus**es that have to be modeled as component bridges with direct interfaces or as a hierarchical component.

objects: XML descriptions of the following types: **catalog**s, **component**s, **design**s, **busDefinition**s, **abstractionDefinition**s, **abstractor**s, **designConfiguration**s, and **generatorChain**s. 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**.

phantom port: A port that only exists in an IP-XACT component; it does not exist on any HDL or TLM component.

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 connection**s or by including them in **bus interface**s connected together by **interconnection**s.

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 (**SCR**s): Additional rules applied to an XML description that cannot be expressed in the **schema**. Typically, these are rules between elements in multiple XML descriptions.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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

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

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

transactional port: A **port** that has an initiative and a kind or a type definition (which can specify the data type of the **port**). 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.

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

NOTE—See 6.11.2.

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

NOTE—See 6.11.6.

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**.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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

XSLT: XSL Transformations is a language for transforming eXtensible Markup Language (XML) documents into other XML documents.

3.2 Acronyms and abbreviations

AHB	AMBA® high-speed bus[15]
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)
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
WSDL	Web Services Description Language
XSLT	Extensible Stylesheet Language Transformations
XML	eXtensible Markup Language
3MD	three levels of meta-data

[15]AMBA 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.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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 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 as *fixed IP* or *configurable IP*.

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).

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

- 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.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

5. Interface definition descriptions

5.1 Definition descriptions

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

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

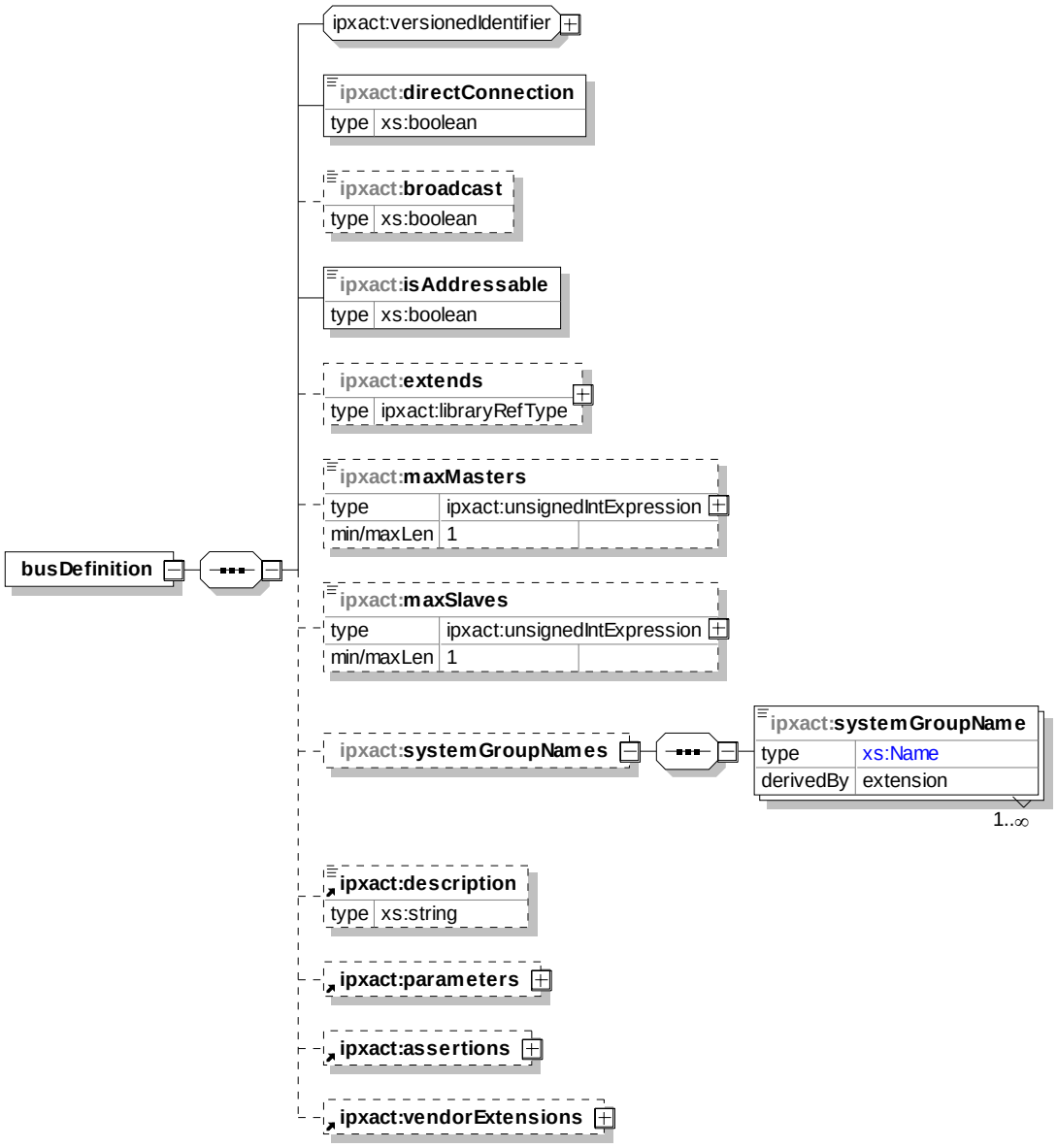
The *abstraction definition* contains the low-level attributes of the interface, including items such as the name, direction, and width of the ports. This is a list of logical ports that may appear on a bus interface for that bus type. Multiple abstractions definitions can be associated with a single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

5.2 Bus definition

5.2.1 Schema

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

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



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.

a) The ***versionedIdentifier*** group provides a unique identifier; it consists of four subelements for a top-level IP-XACT element. See C.25.

b) **directConnection** (mandatory; type: ***boolean***) specifies what connections are allowed. A value of **true** specifies these interfaces may be connected in a direct master to slave fashion. A value of **false** indicates only non-mirror to mirror type connections are allowed (i.e., master–mirroredMaster, slave–mirroredSlave, or system–mirroredSystem).

c) **broadcast** (optional; type: ***boolean***; default: **false**) indicates this bus definition supports *broadcast mode*, which means bus interfaces using this definition support one-to-many interface connections.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

d) **isAddressable** (mandatory; type: ***boolean***) specifies the bus has addressing information. A value of **true** specifies these interfaces contain addressing information and a memory map can be traced through this interface. A value of **false** indicates these interfaces do not contain any traceable addressing information. See also: 6.3.

e) **extends** (optional; type: ***libraryRefType*** (see C.11)) specifies if this definition is an extension from another bus definition. See also: 5.11.

f) **maxMasters** (optional; type: ***unsignedIntExpression*** (see C.3.7)) specifies the maximum number of masters that can appear in a **channel** element containing bus interfaces using this bus definition. If the **maxMasters** element is not present, the number of allowed masters is unbounded.

g) **maxSlaves** (optional; type: ***unsignedIntExpression*** (see C.3.7)) specifies the maximum number of slaves that can appear in a **channel** element containing bus interfaces using this bus definition. If the **maxSlaves** element is not present, the number of allowed slaves is unbounded.

h) **systemGroupNames** (optional) defines an unbounded list of **systemGroupName** (mandatory; type: ***Name***) elements, which in turn define the possible group names to be used under an **onSystem** element in an abstraction definition. The definition of the group names in the bus definition allows multiple abstraction definitions to indicate which system interfaces match each other. The **systemGroupName** shall be unique with the containing **systemGroupNames** element.

i) **description** (optional; type: ***string***) allows a textual description of the interface.

j) **parameters** (optional) describes any **parameter** that can be used to configure or hold information related to this bus definition. See C.18.

k) **assertions** (optional) contains a list of expressions defining the allowed parameter values. See C.2.

l) **vendorExtensions** (optional) contains any extra vendor-specific data related to the interface. See C.24.

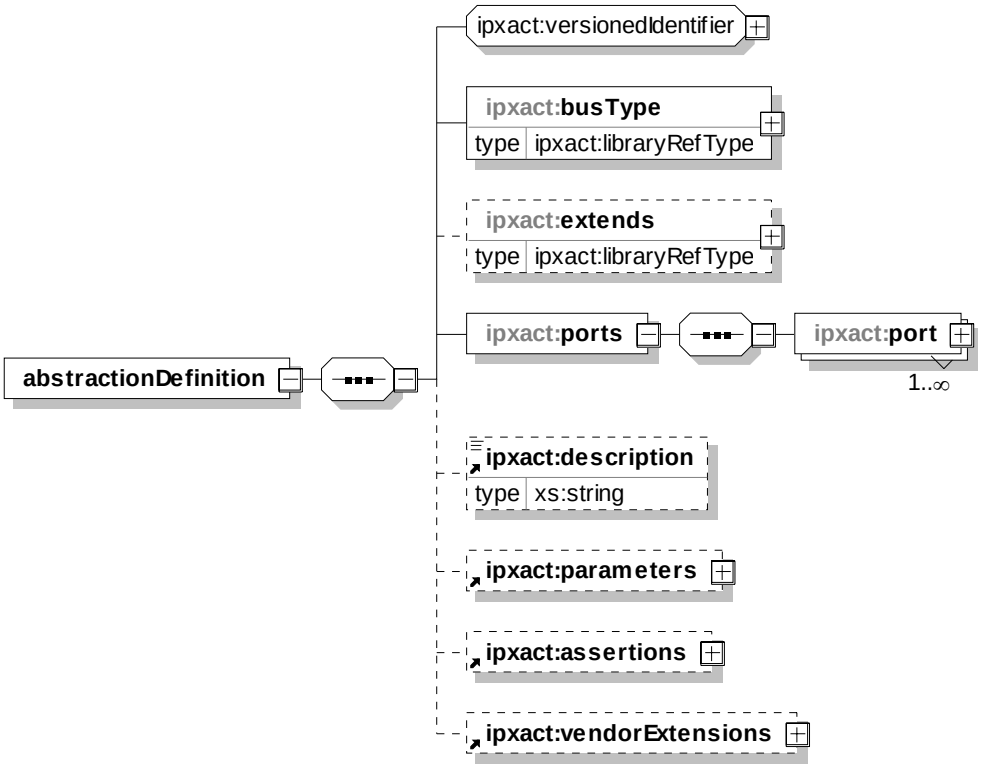
See also: SCR 1.3, SCR 1.10, SCR 1.12, SCR 2.17, SCR 3.18, and SCR 6.16.

5.3 Abstraction definition

5.3.1 Schema

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

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



5.3.2 Description

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

a) The ***versionedIdentifier*** group provides a unique identifier; it consists of four subelements for a top-level IP-XACT element. See C.25.

b) **busType** (mandatory; type: ***libraryRefType*** (see C.11)) specifies the bus definition that this abstraction defines. See also: 5.11.

c) **extends** (optional; type: ***libraryRefType*** (see C.11)) specifies if this definition is an extension from another abstraction definition. The extending abstraction definition may change the definition of logical ports, add new ports, or mark existing logical ports illegal (to disallow their use). See also: 5.11.

d) **ports** (mandatory) is a list of logical ports, see 5.4.

e) **description** (optional; type: ***string***) allows a textual description of the interface.

f) **parameters** (optional) describes any **parameter** that can be used to configure or hold information related to this bus definition. See C.18.

g) **assertions** (optional) contains a list of expressions defining the allowed parameter values. See C.2.

h) **vendorExtensions** (optional) contains any extra vendor-specific data related to the interface. See C.24.

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

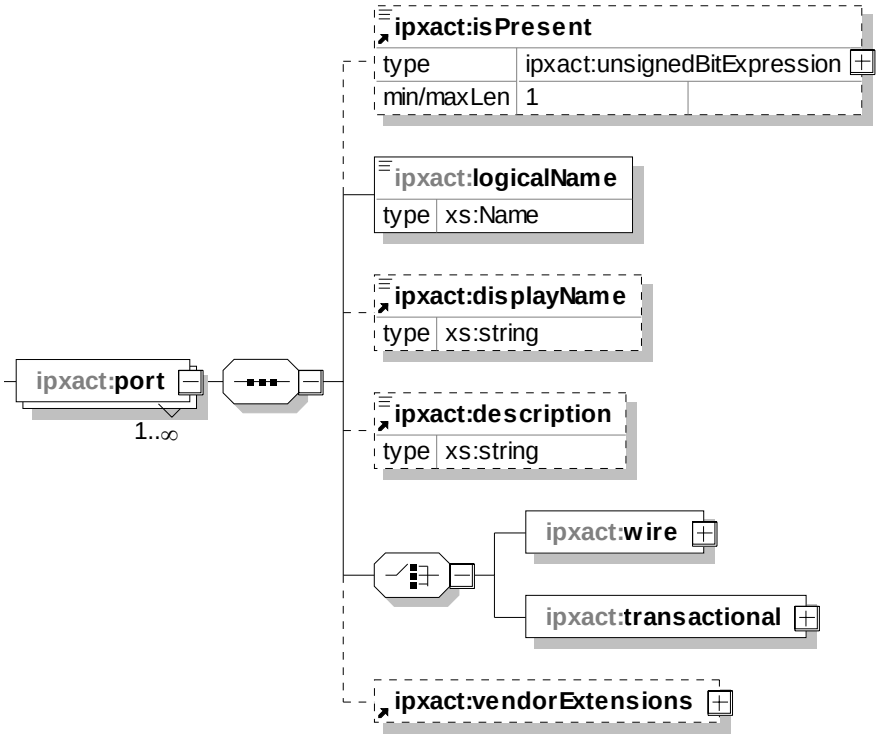
Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

See also: SCR 1.10, SCR 1.12, SCR 1.14, SCR 3.1, SCR 3.15, SCR 3.16, and SCR 6.11.

5.4 Ports

5.4.1 Schema

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



5.4.2 Description

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

a) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

b) **logicalName** (mandatory; type: ***Name***) gives a name to the logical port that can be used later in component description when the mapping is done from a logical abstraction definition port to the components physical port. The **logicalName** shall be unique within the **abstractionDefinition**.

c) **displayName** (optional; type: ***string***) allows a short descriptive text to be associated with the port.

d) **description** (optional; type: ***string***) allows a textual description of the port.

e) Each **port** also requires a **wire** element or a **transactional** element to further describe the details about this port. See 5.5 or 5.9, respectively. A **wire** style port is a port that carries logic values or an array of logic values. A **transactional** style port is a port that carries any other type of information, typically used for TLM.

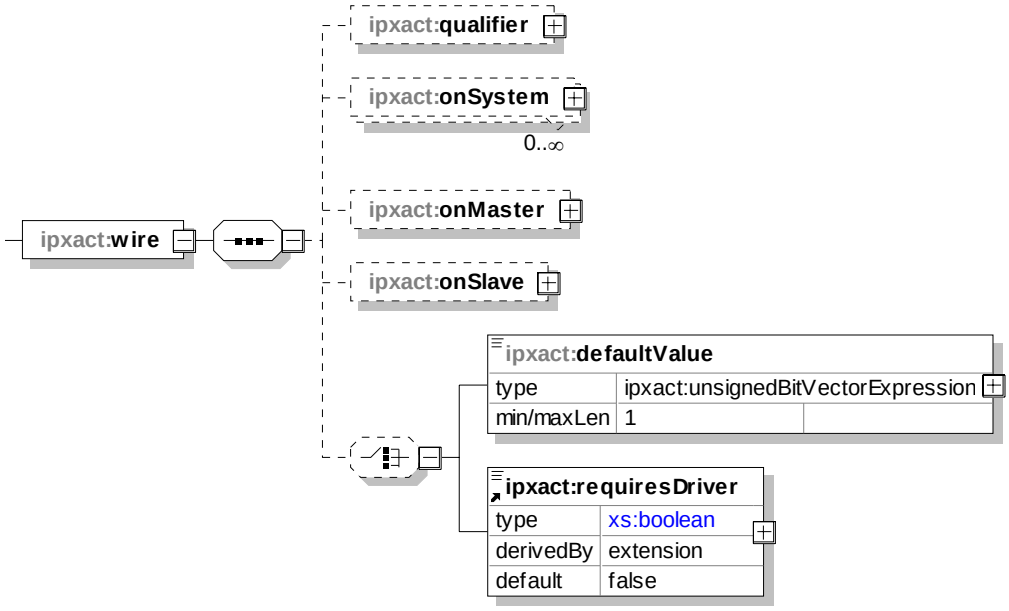
f) **vendorExtensions** (optional) contains any extra vendor-specific data related to the port. See C.24.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

5.5 Wire ports

5.5.1 Schema

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



5.5.2 Description

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

a) **qualifier** (optional) indicates which type of information this wire port carries. See 5.6.

b) **onSystem** (optional) defines constraints, e.g., timing constraints, for this wire port if it is present in a system bus interface with a matching group name.

   1) The **group** (mandatory) attribute indicates the group name for the wire port. It distinguishes between different sets of system interfaces. Usually, all the arbiter ports are processed together, or all the clock or reset ports are processed together. So, this is really a mechanism to specify any sort of non-standard bus interface capabilities for the interconnect. The type of this element is ***Name***.

   2) The group ***wirePort*** specifies what elements are used in this port. See 5.7.

c) **onMaster** (optional) defines constraints for this wire port when present in a master bus interface. The group ***wirePort*** specifies what elements are used in this port. See 5.7.

d) **onSlave** (optional) defines constraints for this wire port when present in a slave bus interface. The group ***wirePort*** specifies what elements are used in this port. See 5.7.

e) A wire may also contain one of the following elements.

   1) **defaultValue** (type: ***unsignedBitVectorExpression*** (see C.3.6)) contains the default logic value for this wire port. This value is applied when the **busInterface** is connected, this logical port is not connected, and there is not an ad hoc connection to the corresponding physical port.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

2) **requiresDriver** (type: ***boolean***; default: **false**) specifies whether the port requires a driver when used in a completed design. If this element is not present, its effective value is **false**, indicating this port does not require a driver. When set to **true**, the attribute **driverType** further qualifies what driver type is required: **any** (meaning any logic signal or value), **clock** (meaning a repeating type waveform), or **singleshot** (a non-repeating type waveform). If **driverType** is not present, its effective value is **any**.

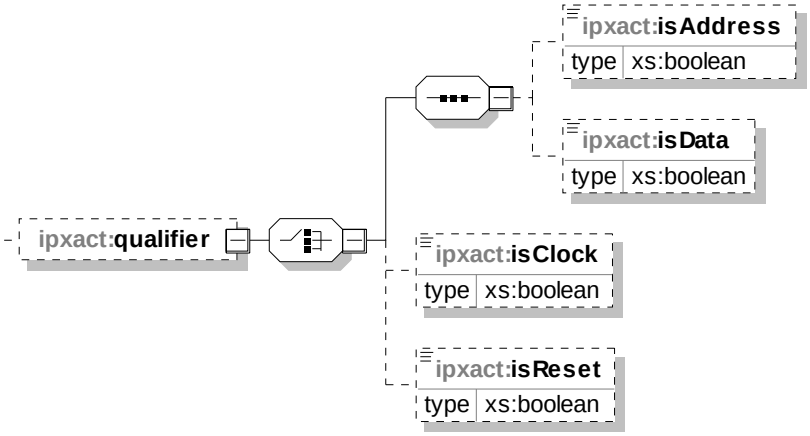
NOTE—The **onMaster**, **onSlave**, and **onSystem** elements associated with each logical port provide optional constraints. If any of these are missing, there are no constraints for how the port appears on interfaces with that mode (master, system, or slave). A port may appear in any system interface 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 6.14.

5.6 Qualifiers

5.6.1 Schema

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



5.6.2 Description

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

a) **isAddress** (optional; type: ***boolean***), when **true**, specifies the port contains address information. This **qualifier** may be paired with the **isData** element (e.g., used with serial protocols). See also: Clause 12.

b) **isData** (optional; type: ***boolean***), when **true**, specifies the port contains data information. This data resides in registers defined in the memory map referenced by the interface. The width defined by the port on each side of the two connected bus interfaces can be used to determine which portions of the data may be lost or gained (tied off to defaults) during transfers if the two widths do not match. This **qualifier** may be paired with the **isAddress** element (e.g., used with serial protocols). See also: Clause 12.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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

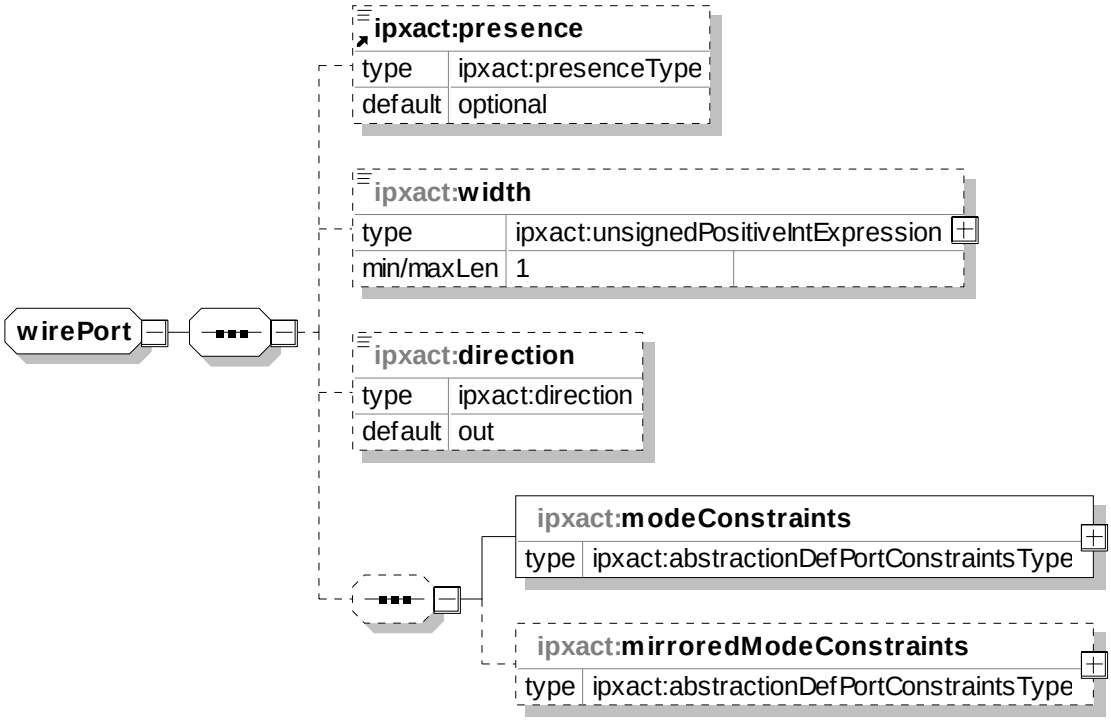
d) **isReset** (optional; type: ***boolean***), when **true**, specifies this port is a reset for this bus interface., i.e., it provides the necessary input to put the interface into a known state. No method of processing is implied with this tag. This tag should only be applied to pure reset ports.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

See also: SCR 6.16, SCR 9.1, SCR 9.2, and SCR 12.8.

5.7 Wire port group

5.7.1 Schema

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



5.7.2 Description

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

a) **presence** (optional; default: **optional**) provides the capability to require or forbid a port from appearing in a **busInterface**. The three possible values are **illegal**, **required**, or **optional**. If this element is not present, its effective value is **optional**.

b) **width** (optional; type: ***unsignedPositiveIntExpression*** (see C.3.9)) represents the number of logical bits that are required to represent this port. When mapping to this logical port in a **busInterface**/

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

portmap, the numbering shall start from `0` to `width-1`. If **width** is not specified, the component shall define the number of bits in this port, but the logical portmap numbering shall still start at `0`. If necessary, logical bit `0` shall be the least significant bit.

c) **direction** (optional; default: **out**) restricts the direction of the port relative to the non-mirrored interface (see 6.2). The three possible values are **in**, **out**, or **inout**.

d) Each ***wirePort*** group can also have a sequence of **modeConstraints** and **mirroredModeConstraints** specifying the default synthesis constraints associated with this logical port. The **modeConstraints** apply to this port if it appears in a non-mirrored *mode* bus interface (see 5.8). Any **mirroredModeConstraints** apply to this port if it appears in a mirrored-*mode* bus interface (see 5.8).

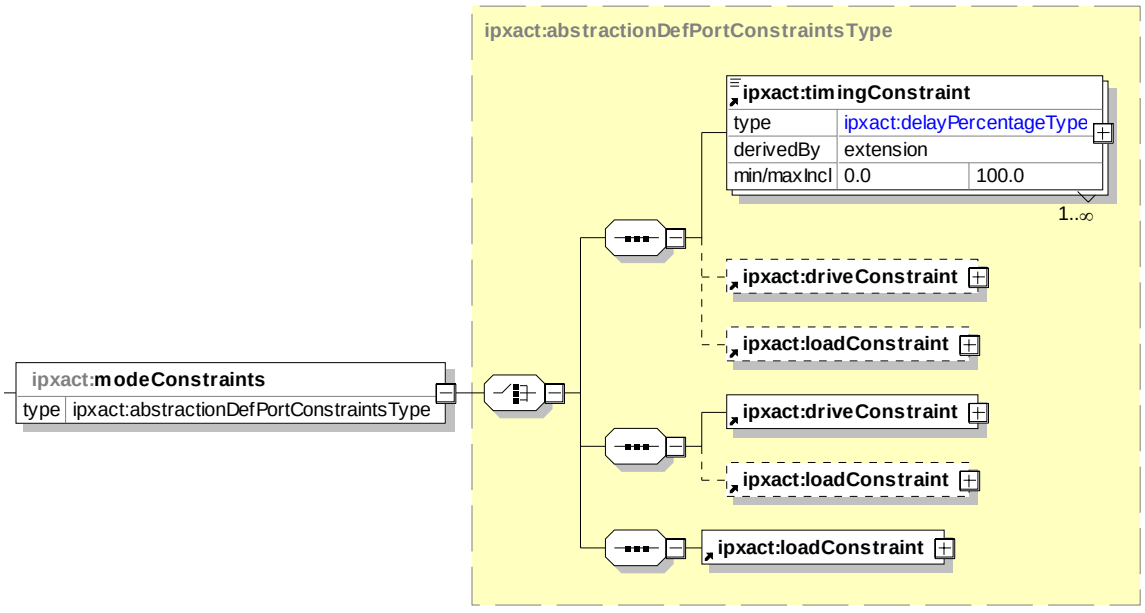
   If **mirroredModeConstraints** are not specified, the **modeConstraints** also apply to this port in a mirrored-*mode* bus interface.

See also: SCR 6.5, SCR 6.6, SCR 6.7, and SCR 6.17.

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

5.8.1 Schema

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



5.8.2 Description

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

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

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

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

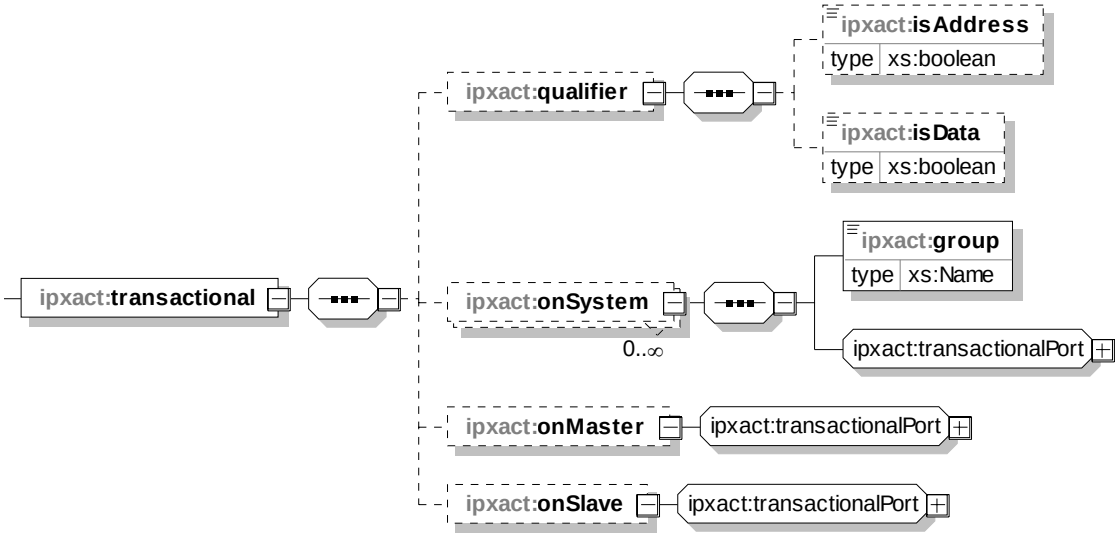
c) **loadConstraint** (optional) element defines a technology-independent load constraint associated with the containing wire port. See 6.12.14.

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

5.9 Transactional ports

5.9.1 Schema

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



5.9.2 Description

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

a) The **qualifier** (optional) element indicates which type of information this transactional port carries. It contains either or both of the following elements.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

1) **isAddress** (optional; type: ***boolean***) specifies the port contains address information.
2) **isData** (optional; type: ***boolean***) specifies the port contains data information.

b) **onSystem** (optional) defines constraints for this transactional port if it is present in a system bus interface with a matching group name.
1) The **group** (type: ***Name***) attribute indicates the group name for the transactional port. It distinguishes between different sets of system interfaces. Usually, all the arbiter ports are processed together, or all the clock or reset ports are processed together. So, this is really a mechanism to specify any sort of non-standard bus interface capabilities for the interconnect. The **group** name shall match the one specified in the bus definition **systemGroupName**.
2) The group ***transactionalPort*** specifies what elements are used in this port. See 5.10.

c) **onMaster** (optional) defines constraints for this transactional port when present in a master bus interface. The group ***transactionalPort*** specifies what elements are used in this port. See 5.10.

d) **onSlave** (optional) defines constraints for this transactional port when present in a slave bus interface. The group ***transactionalPort*** specifies what elements are used in this port. See 5.10.

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

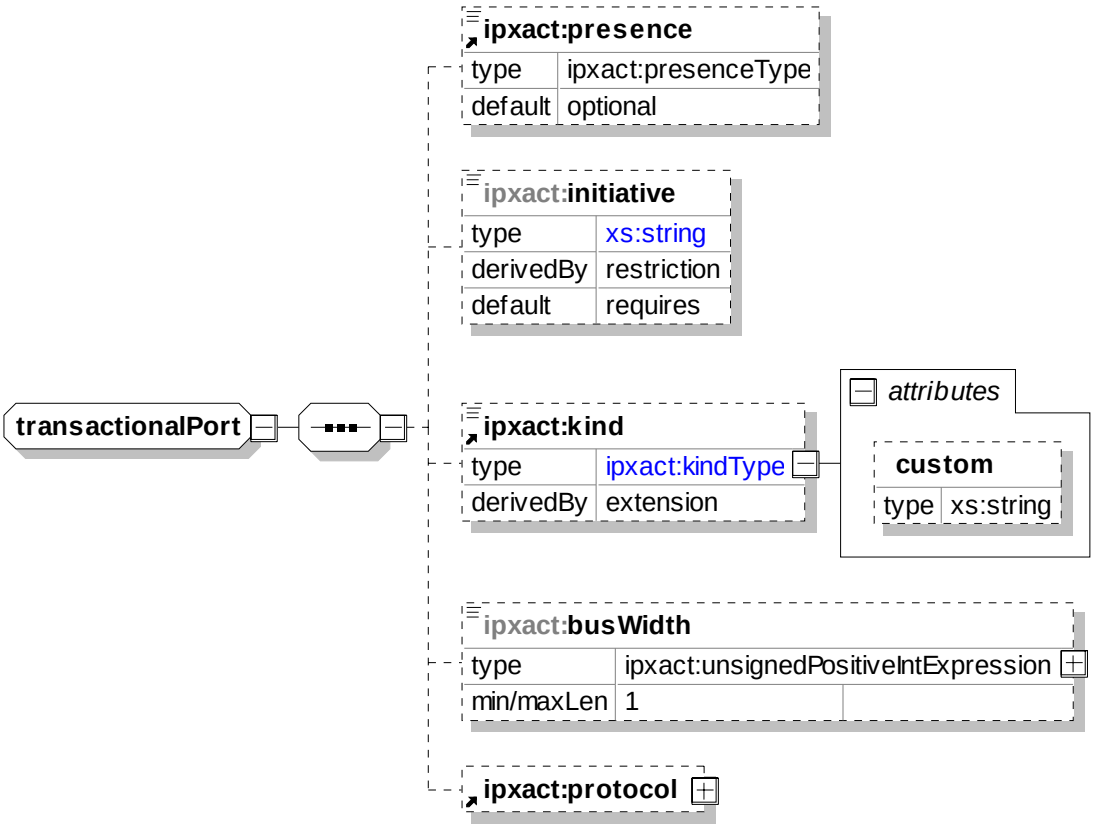
See also: SCR 6.13, SCR 6.14, and SCR 6.16.

5.10 Transactional port group

5.10.1 Schema

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

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



5.10.2 Description

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

a) **presence** (optional; default: **optional**) provides the capability to require or forbid a port to appear in a **busInterface**. Its three possible values are **illegal**, **required**, or **optional**. If this element is not present, its effective value is **optional**.

b) **initiative** (optional; type: ***string***; default: **requires**) defines the type of access: **requires**, **provides**, or **both**. For example, a SystemC `sc_port` is defined using **requires**, since it requires a SystemC interface.

c) **kind** (optional) specifies the transactional port's type. It can take one of the following `enum` values:

tlm_port, **tlm_socket**, **simple_socket**, **multi_socket**, or **custom**. When **custom**, a *name* can be specified in the custom attribute. Also, the **tlm_port** should only be used for TLM1 notation; the other `enum` values should be used for TLM2 sockets.

d) **busWidth** (optional; type ***unsignedPositiveIntExpression*** (see C.3.9)) specifies the data width in bits, e.g., `32` or `64`.

e) **protocol** (optional) defines how the information is transported by the transactional port. (see 6.12.18).

See also: SCR 6.2, SCR 6.3, SCR 6.4, SCR 6.8, and SCR 6.17.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

5.11 Extending bus and abstraction definitions

5.11.1 Extending bus definitions

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

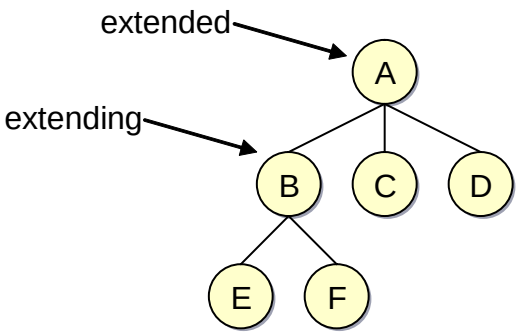


Figure 8—Extends relation hierarchy tree

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

5.11.2 Extending abstraction definitions

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

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

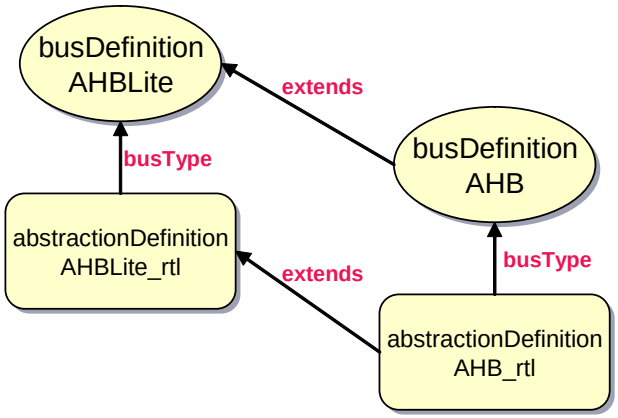


Figure 9—Example of extending

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

See also: SCR 3.16.

5.11.3 Modifying definitions

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

Table 1—Elements of *extending* bus definition

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

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 1—Elements of *extending* bus definition *(continued)*

assertions	Yes	
vendorExtensions	Yes	

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

Table 2—Elements of *extending* abstraction definition

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

The *extending* abstraction definition may add new ports and the *extending* abstraction definition may mark certain ports as illegal to disallow their use. Table 3 specifies which port elements may be modified when extending bus definitions. The *Modifiable* column should be checked after elaboration.

Table 3—Elements of a port in an *extending* abstraction definition

Item	Modifiable	Note
logicalName	No	Changing this name implies a port that is different from the one in the *extended* **abstractionDefinition**.
requiresDriver	Yes	
isAddress	No	
isData	No	
isClock	No	
isReset	No	
onSystem/group	Yes	
presence	Yes	
width	Yes	
direction	No	
modeConstraints	Yes	
mirroredModeConstraints	Yes	

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 3—Elements of a port in an *extending* abstraction definition *(continued)*

defaultValue	Yes	This default can be used to set a value for the extended abstraction definition logical port, if this port is not mapped or its presence is marked as illegal.
initiative	No	
kind	No	
busWidth	No	
protocol	No	
vendorExtensions	Yes	

5.12 Clock and reset handling

Abstraction definitions shall include all the logical ports that can participate in the protocol of the bus; bus interfaces also need to map to the component all the logical ports that participate in the protocol of that bus interface subject to its **presence** (see 5.7.2).

This requirement applies to clock and reset ports as much as it does to other ports. If the protocol of a bus is dependent on a clock or reset port, the abstraction definition for that bus shall include that clock or reset port. Similarly if the bus protocol at a bus interface is dependent on a particular clock or reset port, the port map of that bus interface shall include that port. The clock or reset port, however, does not need to exist as a port of the component implementation, since it may be mapped to a phantom port of the component (see 6.12.20.2). Also, since multiple bus ports may be mapped to a single component port (and component ports may also participate in ad hoc connections), the clock routing is not required to match or be defined by the bus infrastructure.

If the clock and reset ports are associated with the protocol of a bus interface, the special purpose clock or reset **busInterface** (see 6.5.1) can be referenced from any bus interface using the **clockInterfaceRef** and **resetInterfaceRef** subelements.

In some cases, a component may have clock or reset ports that are not associated with and do not participate in the protocol of any bus interface, but do provide a clock or reset to the internal logic of the component instead, e.g., a processor clock. In such cases, the clock port should be included in a special purpose clock or reset bus interface, with an appropriate special purpose bus type, or not be mapped into any interface and connected using ad hoc connections instead.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

6. Component descriptions

6.1 Component

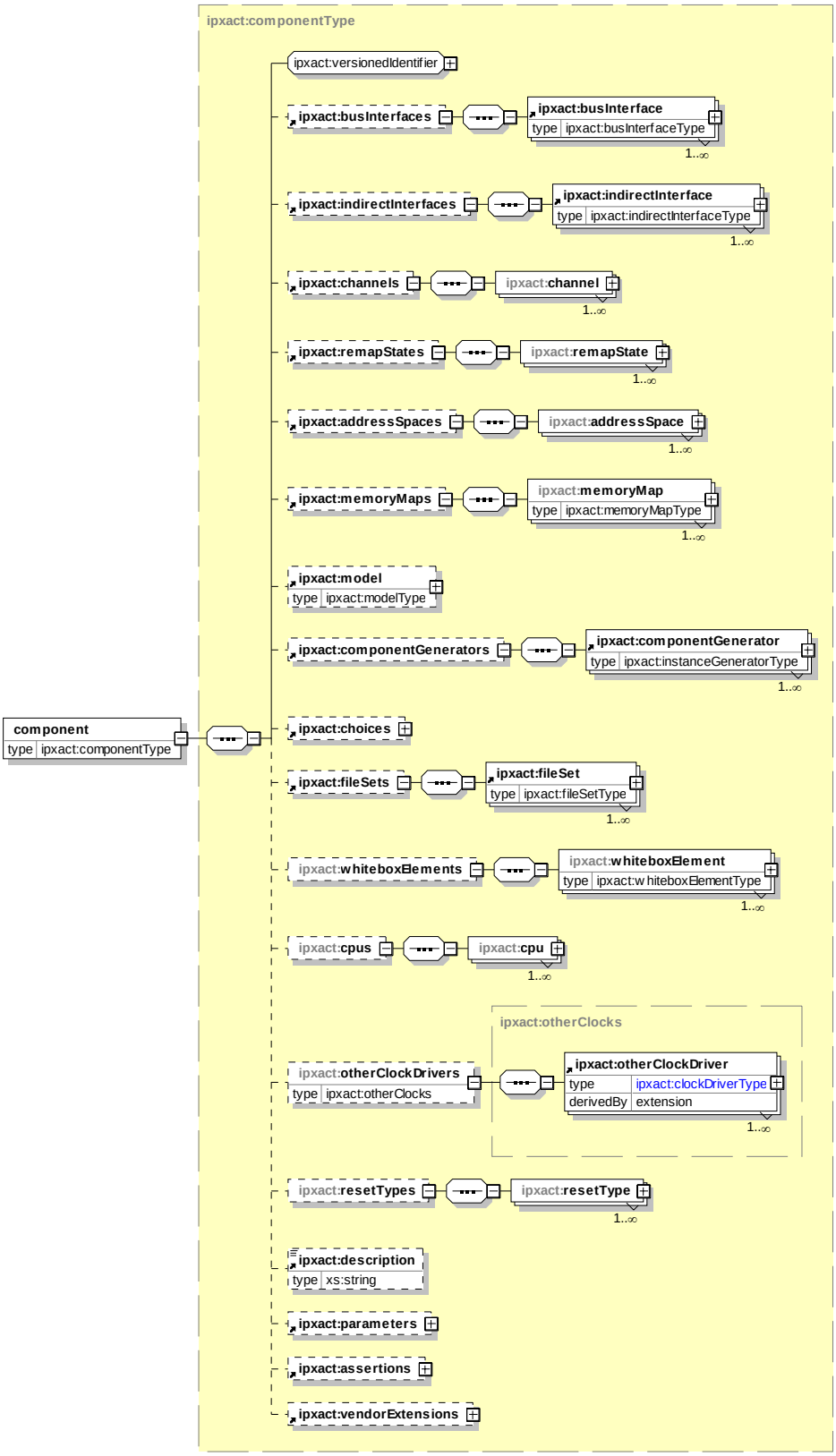
An IP-XACT *component* is used to describe the meta-data associated with any IP that can be instantiated in a design. Examples include IP such as cores (processors, co-processors, DSPs, etc.), peripherals (memories, DMA controllers, timers, UART, etc.), buses (simple buses, multi-layer buses, cross bars, network on chip, etc.), or any other IP block that can be instantiated in a design. An IP-XACT component can be of two kinds: static or configurable. A DE cannot change a *static component*. A *configurable* (or *parameterized*) *component* has configurable elements (such as parameters) that can be configured by the DE and these elements may also configure the RTL or TLM model.

An IP-XACT component can be a hierarchical object or a leaf object. *Leaf components* do not contain other IP-XACT components, while *hierarchical components* contain other IP-XACT sub-components. This can be recursive by having hierarchical components that contain hierarchical components, etc.—leading to the concept of *hierarchy depth*. The IP being described may have a completely different hierarchical arrangement in terms of its implementation in RTL or TLM to that of its IP-XACT description. So, a description of a large IP component may be made up of many levels of hierarchy, but its IP-XACT description need only be a leaf object as that completely describes the IP. On the other hand, some IP can only be described in terms of a hierarchical IP-XACT description, no matter what the arrangement of the implementation hierarchy.

6.1.1 Schema

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

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

6.1.2 Description

Each element of a **component** is detailed in the rest of this subclause; the main sections of a **component** are as follows:

a) ***versionedIdentifier*** group provides a unique identifier; it consists of four subelements for a top-level IP-XACT element. See C.25.

b) **busInterfaces** (optional) specifies all the interfaces for this component. A **busInterface** is a grouping of ports related to a function, typically a bus, defined by a bus definition and abstraction definition. See 6.5.

c) **indirectInterfaces** (optional) specifies access points and access information for any indirect memory maps. See 6.6.

d) **channels** (optional) specifies the interconnection between interfaces inside of the component. See 6.7.

e) **remapStates** (optional) specifies the combination of logic states on the component ports and translates them into a logical name for use by logic that controls the defined address map. See 6.10.2.

f) **addressSpaces** (optional) specifies the addressable area as seen from **busInterface**s with an interface mode of **master** or from **cpu**s. See 6.8.1.

g) **memoryMaps** (optional) specifies the addressable area as seen from **busInterface**s with an interface mode of **slave**. See 6.9.

h) **model** (optional) specifies all the different views, ports, and model configuration parameters of the component. See 6.12.

i) **componentGenerators** (optional) specifies a list of generator programs attached to this component. See 6.13.

j) **choices** (optional) specifies multiple enumerated lists. These lists are referenced by other sections of this component description. See 6.14.

k) **fileSets** (optional) specifies groups of files and possibly their function for reference by other sections of this component description. See 6.15.

l) **whiteboxElements** (optional) specifies all the different locations in the component that can be accessed for verification purposes. See 6.16.

m) **cpus** (optional) indicates this component contains programmable processors. See 6.18.

n) **otherClockDrivers** (optional) specifies any clock signals that are referenced by implementation constraints, but are not external ports of the component. See 6.12.16.

o) **resetTypes** (optional) specifies user-defined reset types referenced within field reset elements. See 6.19.

p) **description** (optional) allows a textual description of the component. The **description** element is of type ***string***.

q) **parameters** (optional) describes any **parameter** that can be used to configure or hold information related to this component. See C.18.

r) **assertions** (optional) contains a list of expressions defining the allowed parameter values. See C.2.

s) **vendorExtensions** (optional) contains any extra vendor-specific data related to the component. See C.24.

See also: SCR 1.10.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

6.2 Interfaces

Each IP component normally identifies one or more bus interfaces. *Bus interfaces* are groups of ports that belong to an identified bus type [i.e., a reference to a **busDefinition** (see 5.2)] and abstraction types [i.e., references to **abstractionDefinition**s (see 5.3)]. The purpose of the bus interface is to map the physical ports of the component to the logical ports of the abstraction definitions. This mapping provides more information about the interface.

There are seven possible modes for a bus interface: a bus interface may be a master, slave, or system interface, and may be direct or mirrored. The seventh interface mode is the monitor mode. A monitor interface can be used to connect IP into the design for verification.

6.2.1 Direct interface modes

A *master interface* is the interface mode that initiates a transaction (like a read or write) on a bus. Master interfaces tend to have *associated address spaces*.

A *slave interface* is the interface mode that terminates or consumes a transaction initiated by a master interface. Slave interfaces often contain information about the registers that are accessible through the slave interface.

A *system interface* is neither a master nor slave interface; this interface mode allows specialized (or non-standard) connections to a bus, such as external arbiters. System interfaces can be used to handle situations not covered by the bus specification or deviations from the bus specification standard.

The following guidelines also apply to the direct interface modes.

— If a port participates in the protocol of the master or slave interfaces, it shall be included in master and slave interfaces. System interfaces often contain some of the same ports as master or slave interfaces.

— Some buses have specialized sideband ports. If these are tied or related to the standard ports in the bus (as opposed to being completely stand-alone), these ports should have some sort of **system** element designator in the bus definition.

6.2.2 Mirrored interface modes

As the name suggests, a *mirrored interface* has the same (or similar) ports to its related direct bus interface, but each port's direction or initiative is reversed. So a port that is an input on a direct bus interface would be an output in the matching mirrored interface. A mirrored bus interface (like its direct counterpart) supports the master, slave, and system interface modes.

6.2.3 Monitor interface modes

A *monitor interface* connects to a master, slave, system, mirrored-master, mirrored-slave, or mirrored-system for observation. The connection shall not modify the connected interfaces. A monitor interface is identified by using the **monitor** element in the interface definition and specifying the type of active interface being monitored (master, slave, etc.).

6.3 Interface interconnections

IP-XACT provides for three different types of connections between interfaces. A *direct connection* is a connection between a master interface and a slave interface. A *mirrored-non-mirrored connection* is a connection between a direct interface and its corresponding mirrored interface (i.e., slave and mirrored-

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

slave). A *monitor connection* is a connection between any interface type (other than monitor) and a monitor interface. It is not possible to connect two mirrored interfaces.

All interconnections are described in a top-level design object. See 7.1.

6.3.1 Direct connection

A direct connection is a connection between a master interface and a slave interface. This connection is typically a point-to-point connection, but may be broadcast (one-to-many) under certain conditions (see 6.3.4). More complex connection schemes with direct connections are also possible with the use of a component containing a **bridge** element(s).

See also: SCR 2.2, SCR 2.11, SCR 2.12, SCR 2.13, and SCR 2.14.

6.3.2 Mirrored-non-mirrored connection

A mirrored-non-mirrored connection is a connection between a master interface and a mirrored-master interface, a slave interface and a mirrored-slave interface, or a system interface and a mirrored-system interface. These connections are typically point-to-point, but may be broadcast (one-to-many) under certain conditions (see 6.3.4). More complex connection schemes with mirrored-non-mirrored connections are also possible with the use of a component containing a **channel** element.

See also: SCR 2.2, SCR 2.13, and SCR 2.14.

6.3.3 Monitor connection

A monitor connection is a connection between a monitor interface and any other interface mode: master, mirrored-master, slave, mirrored-slave, system, or mirrored-system interface. The monitor interface is defined for only one mode and can only be used with that specific mode. Monitor connections are purely for non-intrusive observation of an interface. These connections are single-point to multi-point connections: the single point being the interface to be monitored and the multi-point being the monitor interface. More than one monitor may be attached to the same interface. The monitor connection shall meet the following.

a) The connection of a monitor interface shall not count as a connected interface in the determination of the maximum master or maximum slave calculations.

b) The direction or initiative of ports in a monitor interface cannot be specified in an abstraction definition. All wire ports on a monitor interface shall be treated as having a logical direction of **in**. A monitor interface connected to any active interface shall see the values on the wire ports of the active interface as inputs on its ports regardless of the direction they have on the active interface. All transactional ports on a monitor interface shall be treated as having a logical initiative of **provides**.

See also: SCR 2.2, SCR 4.6, and the SCRs in Table B.4.

6.3.4 Broadcast connections

IP-XACT interface connections are typically point-to-point, but broadcast (one-to-many) interface connections are allowed when the following conditions are met.

a) The **busDefinition** associated with the connected interfaces has the **broadcast** element set to `true`.

b) For each referenced logical port, the connections implied by connecting the associated physical ports so not result in the creation of a signal with multiple drivers.

See also: SCR 2.17.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

6.3.5 Interface logical to physical port mapping

An interface on a component can have multiple port-maps per abstraction-type to associate the physical ports on the component with the logical ports in the abstraction definition. This mapping is what provides the extra information needed to enable a higher level of design.

A *physical port* defined in a component is assigned a physical port name and optionally can be assigned a **left** and a **right** element to represent a vector. The **left** element indicates the first boundary, the **right** element, the second boundary. **left** may be larger than **right** and that **left** may be the MSB or LSB (the **right** being the opposite). The **left** and **right** elements are the (bit) rank of the left-most and right-most bits of the port.

A *logical port* defined in an abstraction definition is assigned a logical port name and, optionally, a width. The logical port is assigned a numbering from `width-1` down to `0` if the width is present. If the width is not present, the logical port number shall have a lower bound of `0` and does not have an upper bound.

6.3.5.1 Mapping rules

Mapping rules describe the assignment of logical bit numbers to physical bit numbers.

a) First, apply all the rules defined in B.1.7 to determine the logical and physical ranges.

b) The mapping is logical.left-> physical.left down to logical.right-> physical.right.

6.3.5.2 Physical interconnections

With all logical bits having been assigned from the abstraction definition to physical port, it is a simple matter to describe the physical connections that result from an interface connection. All connections are made purely based on the logical bit assignment. Like logical bit numbers from each interface are connected. The alignment is always such that logical bit `0` from interface `A` connects to logical bit `0` from interface `B`, logical bit `1` from interface `A` connects to logical bit `1` from interface `B`, and so on.

6.4 Complex interface interconnections

There are two constructs used to connect interfaces of standard components together (traditional components, usually with *master* and *slave* interfaces), a channel and a bridge. These constructs are encapsulated into components. Not only does the **channel** or **bridge** component provide a connection between standard components, but it also provides information on the addressing and data flow. With this information, it is possible to construct things such as a memory map for the system.

A *channel* identifies interfaces in a component that connect a component's master, slave, and system interfaces on the same bus. All masters connected to a channel see each slave at the same physical address. On a channel, only one master may initiate transactions at a time. This does not preclude bus protocols that utilize pipelining or out-of-order completion. A bus that has addresses that are simultaneously seen differently from different masters or a bus that allows transactions from different masters to be simultaneously initiated may only be represented using bus bridges, not channels.

A *bridge* is an interface between two separate buses, which may be of the same or different types. Such a component has at least one master interface (onto the peripheral bus) and one slave interface (onto the main system bus). Crossbar bus infrastructure (e.g., an ARM Multilayer AMBA) is also treated as a component containing bus bridges—such examples might have multiple master and multiple slave interfaces.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

6.4.1 Channel

A *channel* is a general name that denotes the collection of connections between multiple internal bus interfaces. The memory map between these connections is restricted so that, for example, a generator can be called to automatically compute all the address maps for the complete design. A channel can represent a simple wiring interconnect or a more complex structure such as a bus.

A channel also encapsulates the connection between master and slave components. A channel is the construct, which represents the bus infrastructure and allows transactions initiated by a master interface to be completed by a slave interface.

The following rules apply for using channels.

a) A slave connected to a channel has the same address as seen from all masters connected to this channel. Therefore, the slave addresses (as seen by each master) are consistent for the system. As a consequence, all slave interfaces connected to a channel see the same address (if they do not, they are connected to different channels).

b) A channel supports memory mapping and remapping (see 6.9, 6.10, Clause 10, and H.3).

See also: SCR 3.2.

6.4.2 Bridge

Some buses can be modeled using a component as a bridge. A *bridge* is a component that physically links one or more master bus interfaces to a slave bus interface and logically connects the master address space(s) to a slave memory map having two bus types on each side. This component has at least one master bus interface and at least one slave bus interface, each for different protocols, and the bridge translates any signals between them. The slave bus's interface definition References a memory-map which contains **subspaceMap**s (references master-interfaces (see 6.9.9)) or the definition contains a **transparentBridge** element (or a set of them (see 6.5.4.2)) to designate the corresponding master bus interface(s). There are two different types of bridges defined in IP-XACT: transparent and opaque. See also: Annex H.

The bridge relationship is *transparent* (a **transparentBridge** element is used (see 6.5.4.2)) when the address space on the bridged master bus interface is a decoded subset of the main address space, as seen through the bus bridge's slave bus interface. In this case, a slave component connected on the bridged master side shall reserve an address block on the main memory map seen on the bridging slave side. If nothing is attached to the bridged master bus interface, then no address block is reserved on the main memory map.

The bridge relationship is *opaque* (the referenced memory-map contains **subspaceMap**s (see 6.9.9)) when the address space on the bridged master bus interface is not directly accessible to the main address space, as seen from the channel to which the slave bus interface is connected. In this case, the bridging component occupies a single address block, which is the size of its slave bus interface, reserved on the memory map of the masters attached to the main bus channel.

The following rules apply for using bridges.

a) A slave interface can bridge to multiple address spaces. Specifically, a bridge shall have one or more master interfaces and each master interface may have an address space associated with that interface.

b) A bridge can only have direct interfaces. As a consequence, a bridge can directly connect to another component (e.g., master interface to slave interface connection) under the conditions defined in 6.3.1. Or it can connect to a channel (e.g., master interface to mirrored-master interface).

c) A bridge supports memory mapping and remapping (see 6.9, 6.10, Clause 10, and H.3).

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

The transfer of addressing information from the slave interface to the master interface of a bridge is done through the address space assigned to the master interface. This address space defines the visible address range from this master interface.

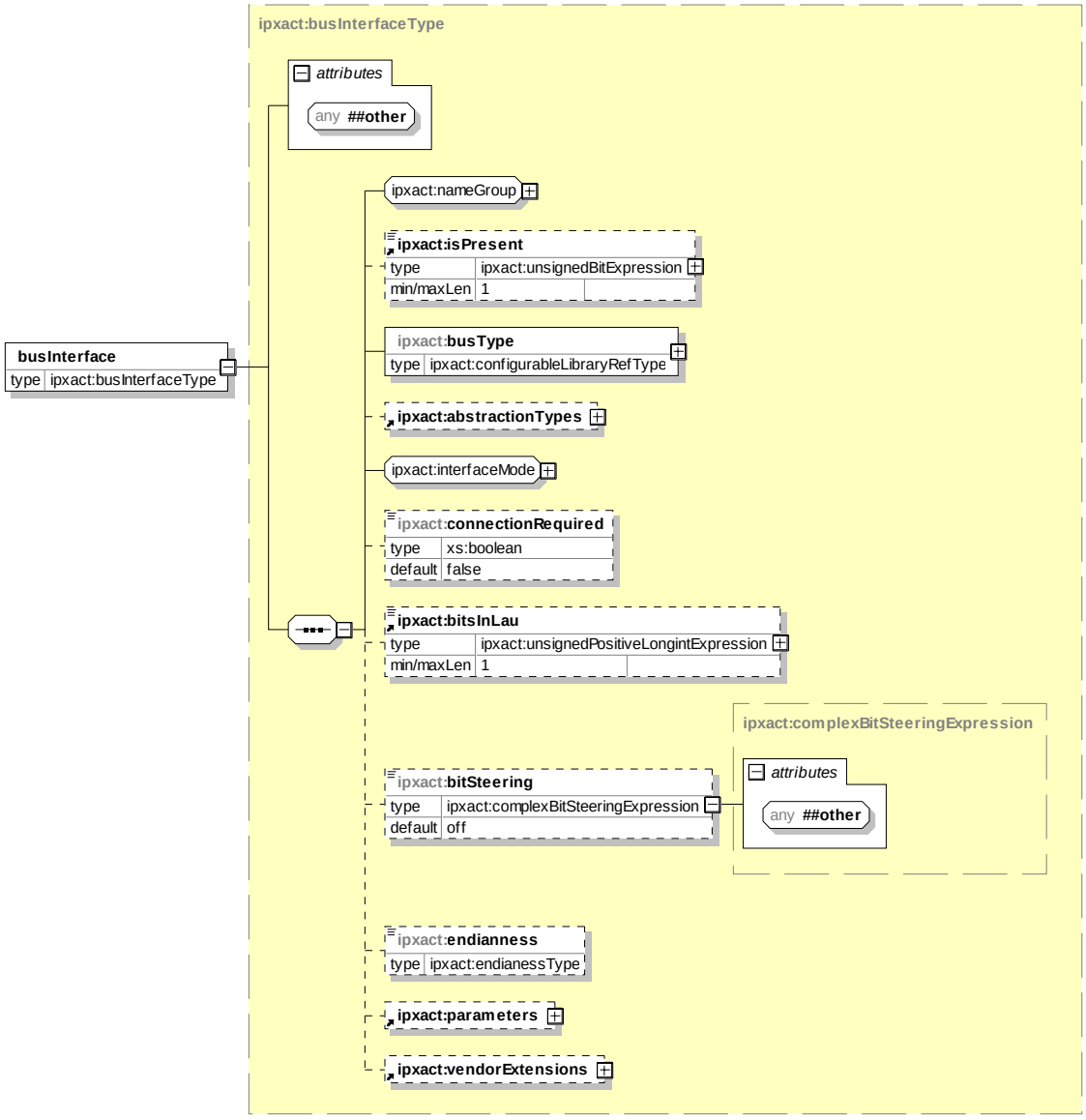
See also: SCR 15.4.

6.5 Bus interfaces

6.5.1 busInterface

6.5.1.1 Schema

The following schema details the information contained in the **busInterfaces** element, which may appear as an element inside the top-level **component** element.



Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

6.5.1.2 Description

Bus interfaces enable individual ports that appear on the component to be grouped together into a meaningful, known protocol. When the protocol is known, a lot of additional information can be written down about the characteristics of that interface.

The **busInterfaces** element contains an unbounded list of **busInterface** elements; therefore, a **component** may have multiple bus interfaces of the same or different types. Each **busInterface** element defines properties of this specific interface in a component. The **busInterface** element also allows for vendor attributes to be applied. It contains the following elements and attributes.

a) ***nameGroup*** group is defined in C.12. The **name** element shall be unique within the containing **component** element.

b) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

c) **busType** (mandatory; type: ***configurableLibraryRefType*** (see C.6)) specifies the bus definition that this bus interface is referenced. A bus definition (see 5.2) describes the high-level attributes of a bus description.

d) **abstractionTypes** (optional) specifies the abstraction definitions of the containing **busInterface**. An abstraction definition describes the low-level attributes of a bus (see 5.3). The **abstractionTypes** element defines a list of **abstractionType** elements. Each abstraction type defines a particular abstraction being used for one or more views and the port mapping associated with the use of that particular abstraction. See 6.5.6.

e) ***interfaceMode*** group describes further information on the *mode* for this interface. There are seven possible modes for an interface: master, slave, mirroredMaster, mirroredSlave, system, mirrored-System and monitor. See 6.5.2.

f) **connectionRequired** (optional; type: ***boolean***; default: **false**), if **true**, specifies when this component is integrated; this interface shall be connected to another interface for the integration to be valid. If **false**, this interface may be left unconnected. If this element is not present, its effective value is **false**.

g) **bitsInLau** (optional; type: ***unsignedPositiveLongintExpression*** (see C.3.10); default: **8**) describes the number of data bits that are addressable by the least significant address bit in the bus interface. It is only appropriate to specify this element for interfaces that are addressable.

h) **bitSteering** (optional; default: **off**) designates if this interface has the ability to dynamically align data on different byte channels on a data bus. This element shall only be specified for interfaces that are addressable. The **bitSteering** element is a choice of two values: **on** indicating this interface uses data steering logic and **off** that this interface does not use data steering logic. The **bitSteering** element is of a type that is either **on**, **off**, or an expression that evaluates to **on** or **off**.

   If these values appear within an expression they are assumed to be parameter ID references. However if they appear as a simple name, they are assumed to be the literal name `on` or `off` with their implied special meanings.

i) **endianness** (optional) indicates the endianness of the bus interface. The two choices are **big** for big-endian and **little** for little-endian. If this element is not present, its effective value is **little**. See also 6.5.1.2.1.

j) **parameters** (optional) specifies any parameter data value(s) for this bus interface. See C.18.

k) **vendorExtensions** (optional) holds any vendor-specific data from other namespaces, which is applicable to this bus interface. See C.24.

See also: SCR 1.4, SCR 2.14, SCR 2.15, SCR 5.24, SCR 5.25, SCR 9.4, SCR 9.5, and SCR 9.6.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

6.5.1.2.1 Endianness

Endianness is defined under the **busInterface** element of the component. There are (only) two legal values (**big** and **little**) to specify the **endianness**.

— *Big endian* (**big**) means the most significant byte of any multi-byte data field is stored at the lowest memory address, which is also the address of the larger field.

— *Little endian* (**little**) means the least significant byte of any multi-byte data field is stored at the lowest memory address, which is also the address of the larger field.

NOTE—The description of endianness is byte-centric as that is the most common least addressable unit (LAU). However, this description generally applies to any size LAU.

6.5.1.2.2 Big-endianness

There are at least two ways for big-endianness to manifest itself, byte-invariant and word-invariant (also known as *middle-endian*); the difference being if data is stored as *word-invariant*, the data is stored differently for transfers larger than a byte, for example,

a) Byte invariant: A word access to address `0x0` is on `D[31:0]`. The MSB is `D[7:0]`, the LSB is `D[31:24]`.

b) Word invariant: A word access to address `0x0` is on `D[31:0]`. The MSB is `D[31:24]`, the LSB byte is `D[7:0]`.

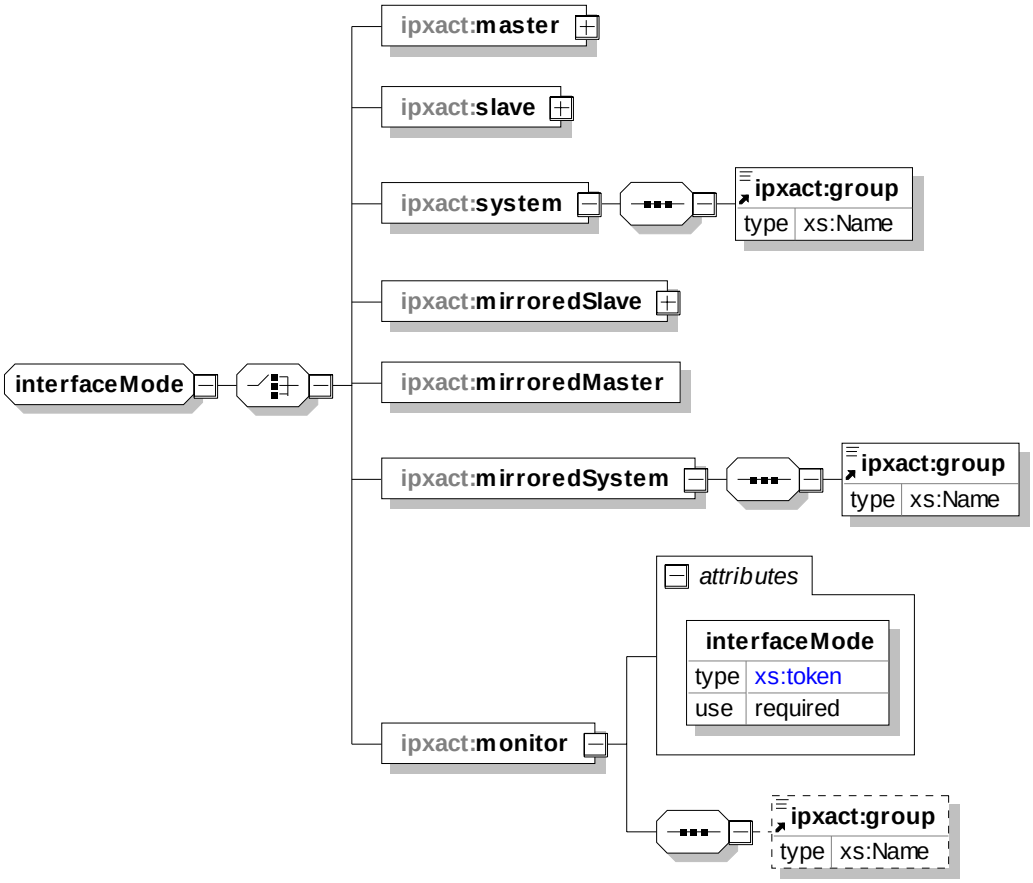
c) In IP-XACT, the interpretation of big-endian is the byte-invariant style.

6.5.2 Interface modes

The following schema details the information contained in the ***interfaceMode*** group, which appears as a group inside the **busInterface** element.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

6.5.2.1 Schema



6.5.2.2 Description

The **busInterface**'s mode designates the purpose of the **busInterface** on this component. There are seven possible modes: three pairs of standard functional interfaces and their mirrored counterparts, and a monitor interface for VIP.

The ***interfaceMode*** group shall contain one of the following elements.

a) A **master** interface mode (sometimes also known as an *initiator*) is one that initiates transactions. See 6.5.3.

b) A **slave** interface mode (sometimes also known as a *target*) is one that responds to transactions. See 6.5.4.

c) A **system** interface mode is used for some classes of interfaces that are standard on different bus types, but do not fit into the master or slave category.

   The **group** (mandatory; type: ***Name***) attribute for the **system** element defines the name of the group to which this system interface belongs.

d) A **mirroredSlave** interface mode is the mirrored version of a slave interface and can provide addition address offsets to the connected slave interface. See 6.5.5.

e) A **mirroredMaster** interface mode is the mirrored version of a master interface.

f) A **mirroredSystem** interface mode is the mirrored version of a system interface.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

The **group** (mandatory; type: ***Name***) attribute for the **mirroredSystem** element defines the name of the group to which this **mirroredSystem** interface belongs.

g) A **monitor** interface mode is a special interface that can be used for verification. This monitor interface mode is used to gather data from other interfaces. See 6.3.3.

1) The **interfaceMode** (mandatory type: ***token***) attribute defines the interface mode for which this monitor interface can be connected: **master**, **slave**, **system**, **mirroredMaster**, **mirroredSlave**, or **mirroredSystem**.

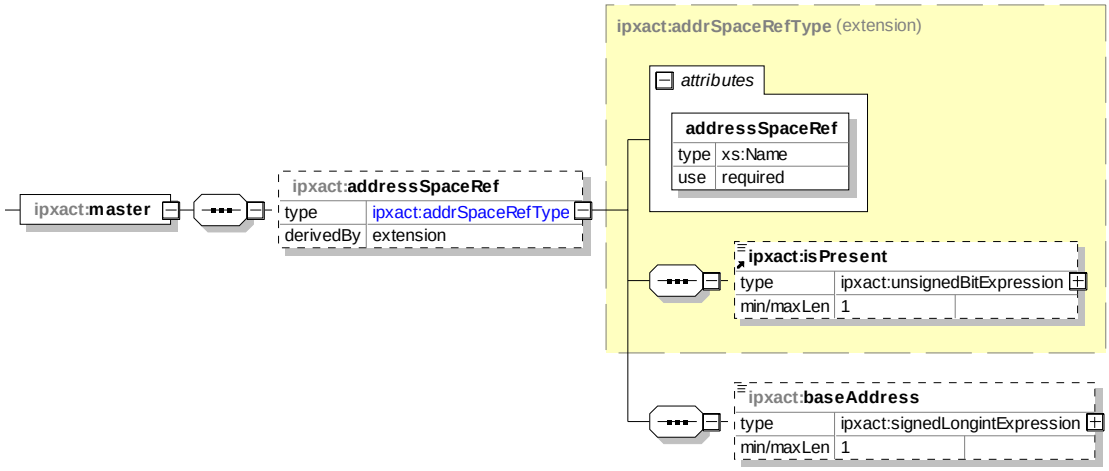
2) The **group** (optional type: ***Name***) element is required if the **interfaceMode** attribute is set to ***system*** or ***mirroredSystem***. This element defines the name of the system group for this monitor interface.

See also: SCR 2.13, SCR 4.3, SCR 4.4, and SCR 6.15.

6.5.3 Master interface

The following schema details the information contained in the **master** element, which appears as an element inside the ***interfaceMode*** group inside **busInterface** element.

6.5.3.1 Schema



6.5.3.2 Description

A **master** interface (also known as an *initiator*) is one that initiates transactions. The **master** element contains the following elements and attributes. **addressSpaceRef** (optional) element contains attributes and subelements to describe information about the range of addresses with which this master interface can generate transactions.

1) **addressSpaceRef** (mandatory; type: ***Name***) attribute references a name of an address space defined in the containing description. The address space shall define the range and width for transaction on this interface. See 6.8.1.2.

2) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

3) **baseAddress** (optional; type: ***signedLongintExpression***, C.3.2)) specifies the starting address of the address space. The address space numbering normally starts at `0`. Some address spaces

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

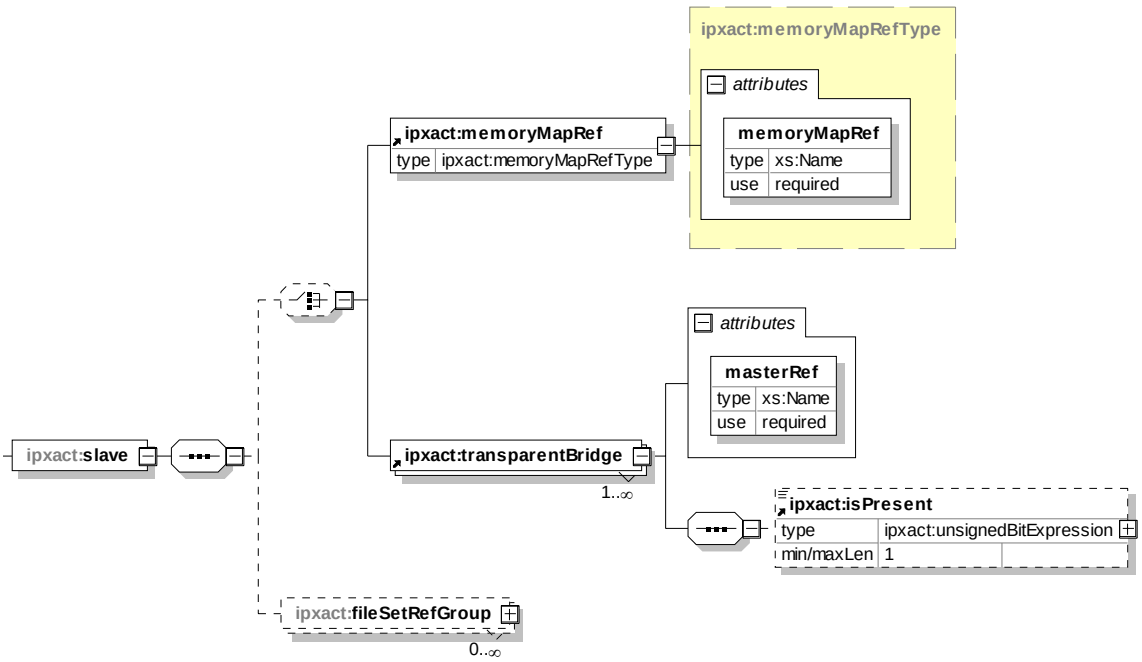
may use *offset addressing* (starting at a number other than `0`) so the base address element can be used to designate this information.

See also: SCR 9.1, SCR 15.7, and SCR 15.14.

6.5.4 Slave interface

The following schema details the information contained in the **slave** element, which appears as an element inside the ***interfaceMode*** group inside **busInterface** element.

6.5.4.1 Schema



6.5.4.2 Description

A **slave** interface (sometimes also known as a *target*) is one that responds to transactions. The memory map reference points to information about the range of registers, memory, or other address blocks accessible through this slave interface. This slave interface can also be used in a bridge application to “bridge” a transaction from a slave interface to a master interface.

a) A **slave** can contain one of the following subelements.

1) **memoryMapRef** contains an attribute that references a memory map.

The **memoryMapRef** (mandatory; type: ***Name***) attribute references a name of a memory map defined in the containing description. The memory map contains information about the range of registers, memory, or other address blocks. See 6.9.

2) **transparentBridge** is an unbounded list of references to master interfaces. If the interface is of a bus definition that is addressable, a **transparentBridge** element may be included. A *transparent bridge* is one in which all addressing entering the slave interface exits the above referenced master interface without any modifications.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

i) The **masterRef** (mandatory; type: ***Name***) attribute shall reference a master interface (see 6.5.3) in the containing description.

ii) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

b) **fileSetRefGroup** (optional) element is an unbounded list of the references to file sets contained in this component. These file set references are associated with this slave interface. This element allows each slave port to reference a unique **fileSet** element (see 6.15) and can further be used to reference a software driver, which can be made different for each slave port.

1) **group** (optional; type: ***Name***) element allows the definition of a group name for the **fileSetRefGroup**.

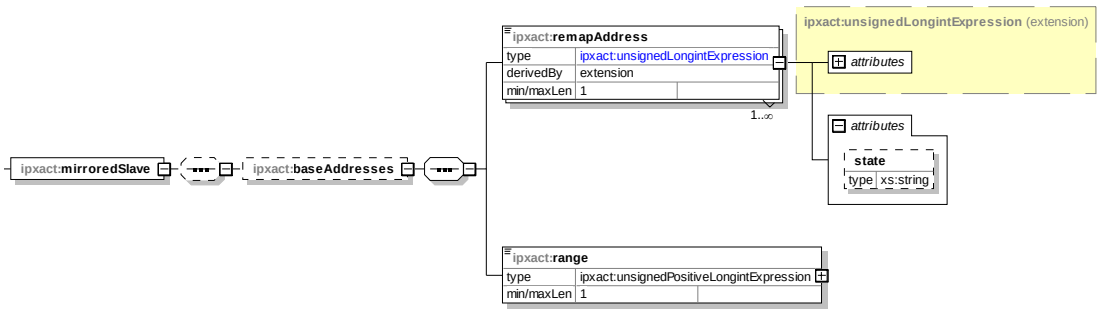
2) **fileSetRef** (optional) is an unbounded list of references to a **fileSet** by **name** within the containing document or another document referenced by the **VLNV**. See C.7.

See also: SCR 3.6, SCR 9.2, and SCR 15.9.

6.5.5 Mirrored slave interface

The following schema details the information contained in the **mirroredSlave** element, which appears as an element inside the ***interfaceMode*** group inside **busInterface** element.

6.5.5.1 Schema



6.5.5.2 Description

A **mirroredSlave** interface is used to connect to a **slave** interface. The **mirroredSlave** interface may contain additional address information in the **baseAddresses** (optional) element.

a) **remapAddress** (mandatory; type ***unsignedLongintExpression***, see C.3.8)) element is an unbounded list that specifies the address offset to apply to the connected slave interface. The **remapAddress** is expressed as the number of addressable units based on the size of an addressable unit as defined inside the containing **busInterface**/**bitsInLau** element.

The **state** (optional; type: ***string***) attribute references a defined state in the component and identifies the **remapState**/**name** for which the **remapAddress** and **range** apply. See 6.10.2.

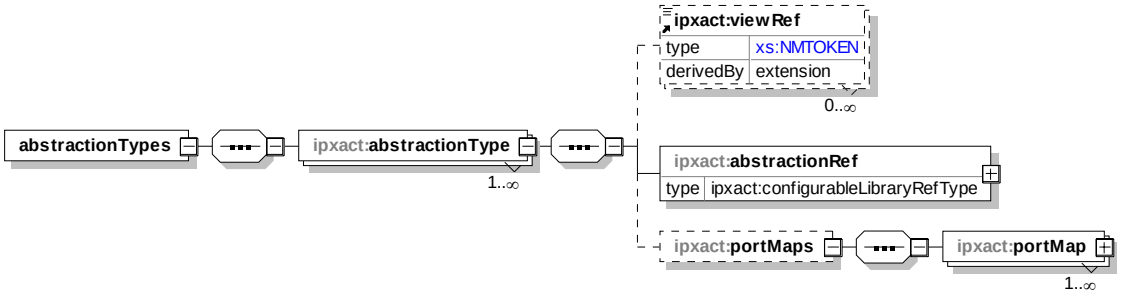
b) **range** (mandatory; type: ***unsignedPositiveLongintExpression*** (see C.3.10)) specifies the address range to apply to the connected **slave** interface.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

6.5.6 Abstraction types

6.5.6.1 Schema

The following schema defines the information contained in the **abstractionTypes** element, which appears within an **abstractorInterface** description.



6.5.6.2 Description

The **abstractionTypes** element specifies the abstraction definition where this bus interface is referenced. Each **abstractionType** contains the following elements.

a) **viewRef** (optional; type: ***NMTOKEN***) specifies the views to which the **abstractionType** applies. See C.26.

b) **abstractionRef** (mandatory; type: ***configurableLibraryRefType*** (see C.6))) is a reference to an abstraction description for this abstractor instance.

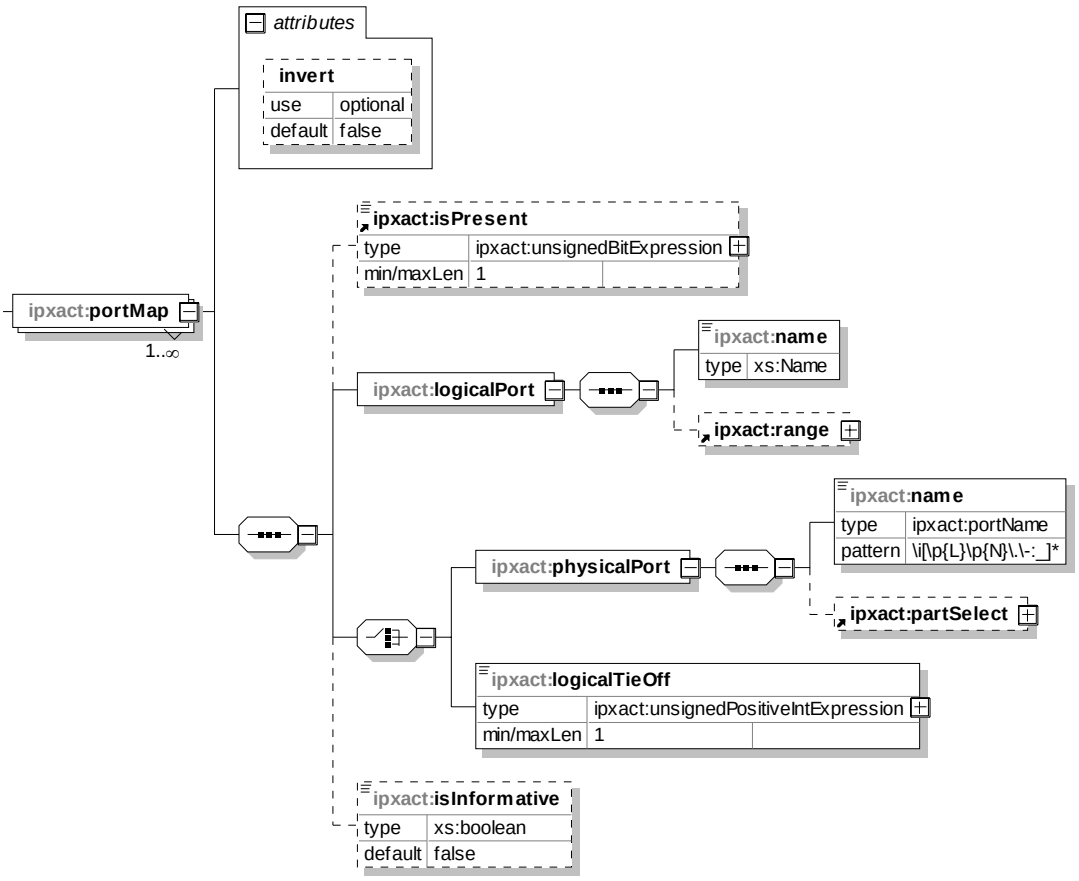
c) **portMaps** (optional) describes the mapping between the abstraction definition's logical ports and the abstractor's physical ports. See 6.5.7.

6.5.7 Port map

The following schema details the information contained in the **portMaps** element, which appears as an element inside **busInterface** element.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

6.5.7.1 Schema



6.5.7.2 Description

Each **portMap** element describes the mapping between the logical ports, defined in the referenced abstraction definition, to the physical ports, defined in the containing component description.

a) **invert** (optional; default: **false**) specifies connections to the indicated physical port shall be logically inverted.

b) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

c) **logicalPort** (mandatory) contains the information on the logical port from the abstraction definition.

   1) **name** (mandatory; type: ***Name***) specifies the logical port name. The name shall be a name of a logical port in the referenced abstraction definition that is defined as legal for this interface mode.

   2) **range** (optional) is used for a vectored logical port to specify the indices of the logical port mapping. See C.22.

d) **portMap** also contains one of the following.

   1) **physicalPort** (type: ***Name***) contains information on the physical port contained in the component.

      i) **name** (mandatory) specifies the physical port name. The name shall be a name of a port in the containing component.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

ii) **partSelect** (optional) is used for a physical port (vectored, arrayed, or both) to specify the indices of the physical port mapping. See C.20.

2) **logicalTieOff** (type: ***unsignedPositiveIntExpression*** (see C.3.9)) indicates the physical connection for this logical port is the specified constant value (instead of a physical port.)

e) **isInformative** (optional; type ***boolean***; default: **false**) when **true**, specifies this **portMap** is only used for information purposes. No connections will be made to the referenced physical port.

The same physical port may be mapped to a number of different logical ports on the same or different bus interfaces, and the same logical port may be mapped to a number of different physical ports. For port mapping rules, see 6.3.5.1.

See also: SCR 6.1, SCR 6.2, SCR 6.3, SCR 6.4, SCR 6.5, SCR 6.6, SCR 6.7, SCR 6.12, SCR 6.13, SCR 6.18, SCR 6.19, SCR 6.20, SCR 6.21, SCR 6.22, SCR 6.23, SCR 15.2, and SCR 15.3.

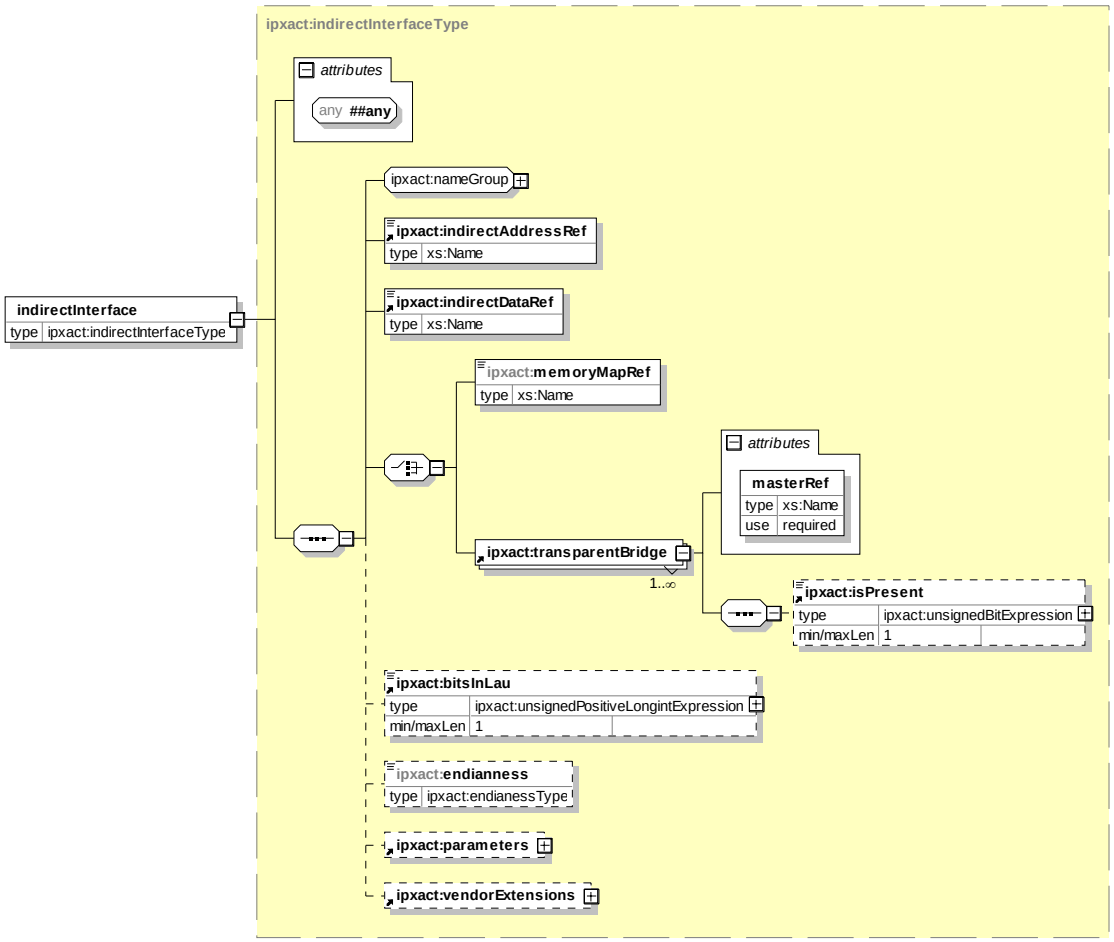
6.6 Indirect interfaces

The contents of an indirectly accessible memory map are not accessible at a fixed address via a bus interface. A bus interface accesses an indirectly accessible memory map via reads and writes to fields the bus interface can directly access.

6.6.1 Schema

The following schema details the information contained in the **indirectInterfaces** element, which may appear as an element inside the top-level **component** element.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



6.6.2 Description

The **indirectInterfaces** element contains an unbounded list of **indirectInterface** elements. Each **indirectInterface** element defines the access details for an indirectly accessible memory map element.

a) ***nameGroup*** group is defined in C.12. The **name** element shall be unique within the containing **indirectInterface** element.

b) **indirectAddressRef** (mandatory; type: ***Name***) references the register field used for addressing the indirectly accessible memory map. The referenced field is usually directly accessible via a bus interface.

c) **indirectDataRef** (mandatory; type: ***Name***) references the register field used for read/write access for the indirectly accessible memory map. The reference is to a unique **fieldID** (see 6.11.8.2). The field is usually accessible via a standard bus interface.

d) An **indirectInterface** element contains either a **memoryMapRef** element or one or more **transparentBridge** elements.

1) **memoryMapRef** (type: ***Name***) references a name of a memory map defined in the containing description. This memory map is indirectly accessible.

2) **transparentBridge** is an unbounded list of references to master interfaces. If the interface is of a bus definition that is addressable, a **transparentBridge** element may be included. A *transparent bridge* is one in which all addressing entering the slave interface exits the above referenced master interface without any modifications.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

i) The **masterRef** (mandatory; type: ***Name***) attribute shall reference a master interface (see 6.5.3) in the containing description.

ii) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

e) **bitsInLau** (optional; type: ***unsignedPositiveLongintExpression*** (see C.3.10); default: **8**) describes the number of data bits that are addressable by the least significant address bit in the bus interface.

f) **endianness** (optional) indicates the endianness of the indirect interface. The two choices are **big** for big-endian and **little** for little-endian. If this element is not present, its effective value is **little**. See also 6.5.1.2.1.

g) **parameters** (optional) specifies any parameter data value(s) for this indirect interface. See C.18.

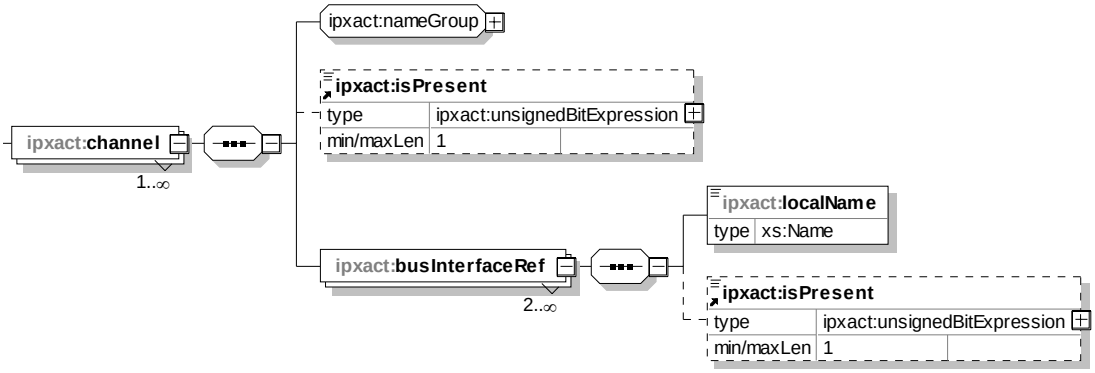
h) **vendorExtensions** (optional) holds any vendor-specific data from other namespaces, which is applicable to this indirect interface. See C.24.

See also: SCR 9.9, SCR 9.10, and SCR 9.11.

6.7 Component channels

6.7.1 Schema

The following schema details the information contained in the **channels** element, which may appear as an element inside the top-level **component** element.



6.7.2 Description

The **channels** element contains an unbounded list of **channel** elements. Each **channel** element contains a list of all the mirrored bus interfaces in the containing component that belong to the same channel.

a) ***nameGroup*** group is defined in C.12. The **name** element shall be unique within the containing **channels** element.

b) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

c) **busInterfaceRef** (mandatory) is an unbound list of references (a minimum of two) to mirrored bus interfaces in the containing component. Each mirrored bus interface in a component may be referenced in any channel at most once. The order of this list may be used by the DE in some way and shall be maintained. See 6.5.1.

1) **localName** (mandatory; type: ***Name***) specifies the name of a mirrored bus interface within this channel.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

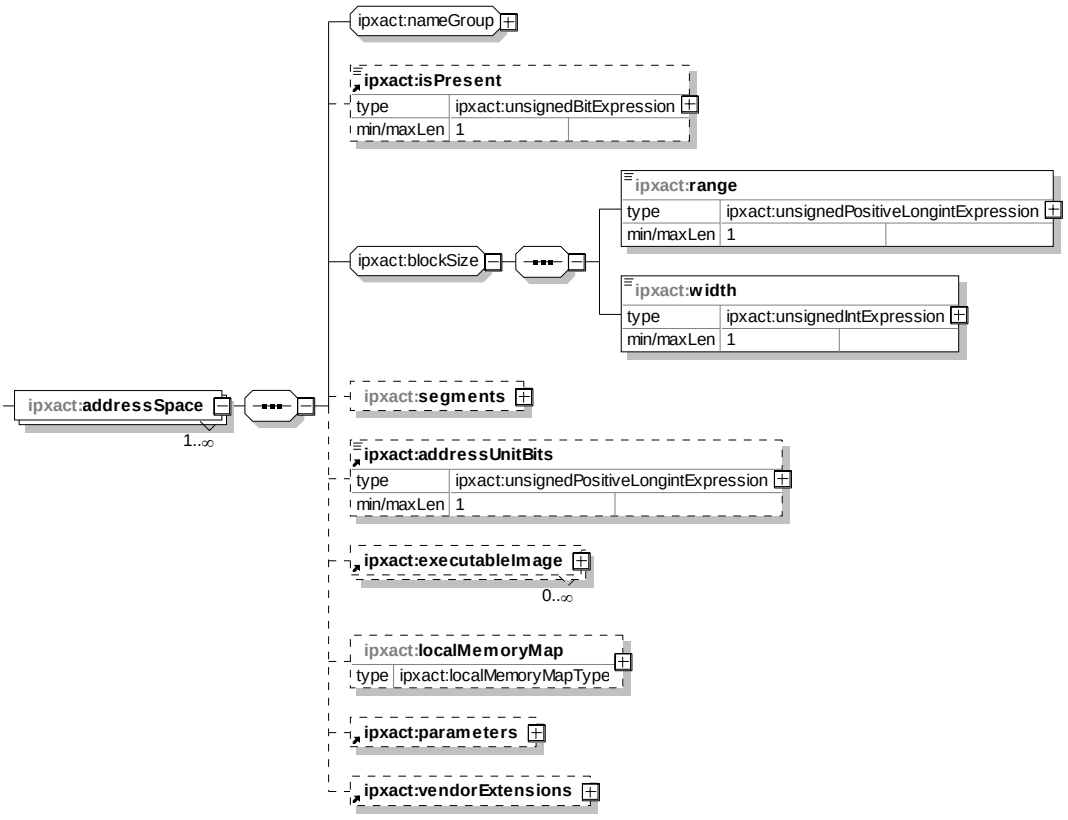
6.8 Address spaces

An address space is defined as a logical addressable space of memory. Each master interface can be assigned a logical address space. Address spaces are effectively the programmer's view looking out from a master interface. Some components may have one address space associated with more than one master interface (for instance, a processor that has a system bus and a fast memory bus). Other components (for instance, Harvard architecture processors) may have multiple address spaces associated with multiple master interfaces—one for instruction and the other for data.

6.8.1 addressSpaces

6.8.1.1 Schema

The following schema details the information contained in the **addressSpaces** element, which may appear as an element inside the top-level **component** element.



6.8.1.2 Description

The **addressSpaces** element contains an unbounded list of **addressSpace** elements. Each **addressSpace** element defines a logical address space seen by a master bus interface. It contains the following elements.

a) ***nameGroup*** group is defined in C.12. The **name** element shall be unique within the containing **addressSpaces** element.

b) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

c) ***blockSize*** group includes the following.

   1) **range** (mandatory; type ***unsignedPositiveLongintExpression*** (see C.3.10)) gives the address range of an address space. See C.22.

   2) **width** (mandatory) is the bit width of a row in the address space. The type of this element is set to **nonNegativeInteger**. The **width** element is of type ***unsignedIntExpression***, see C.3.7.

d) **segments** (optional) describes a portion of the address space starting at an address offset and continuing for a given range. A **segment** can be referenced by a **subspaceMap**. See 6.8.2.

e) **addressUnitBits** (optional; type ***unsignedPositiveLongintExpression*** (see C.3.10)) defines the number of data bits in each address increment of the address space. If this element is not present, it is presumed to be `8`.

f) **executableImage** (optional) describes the details of an executable image that can be loaded and executed in this address space on the processor to which this master bus interface belongs. See 6.8.3.

g) **localMemoryMap** (optional) describes a local memory map that is seen exclusively by this master bus interface viewing this address space. See 6.8.6.

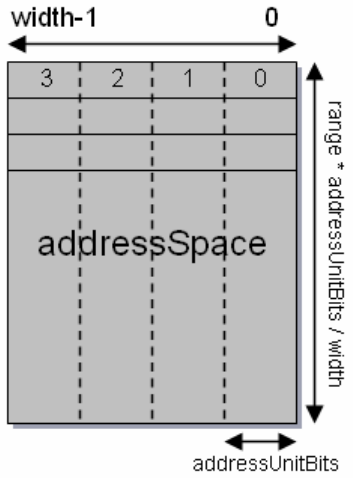
h) **parameters** (optional) specifies any parameter data value(s) for this address space. See C.18.

i) **vendorExtensions** (optional) holds any vendor-specific data from other namespaces, which is applicable to this address space. See C.24.

The **range** and **width** elements are related by the following formulas.

number_of_bits_in_block = `addressUnitBits` × `range`

number_of_rows_in_block = *number_of_bits_in_block* / `width`



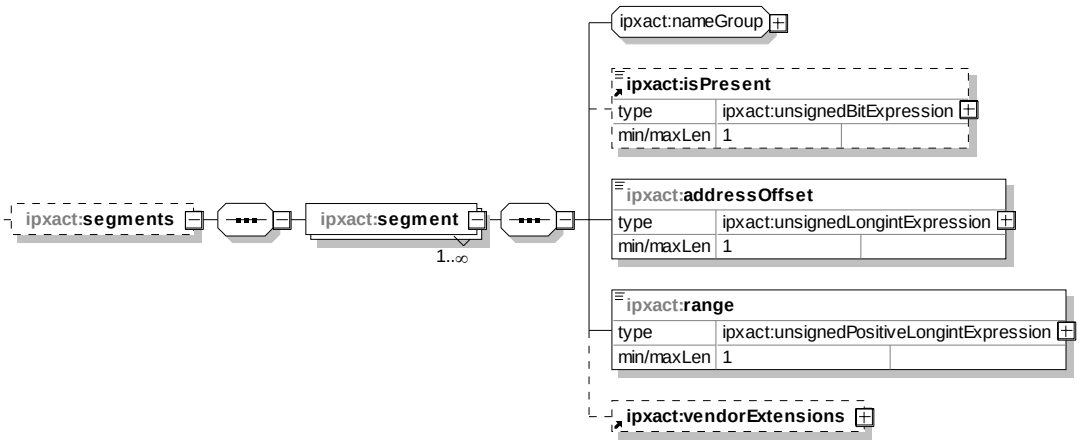
See also: SCR 9.3 and SCR 9.7.

6.8.2 Segments

6.8.2.1 Schema

The following schema details the information contained in the **segments** element, which may appear inside an **addressSpace** element.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



6.8.2.2 Description

The **segments** element contains an unbounded list of **segment** elements. Each **segment** describes the location and size of an area in the containing **addressSpace**.The **segment** element contains the following elements.

a) ***nameGroup*** group is defined in C.12. The **name** element shall be unique within the containing **segments** element.

b) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

c) **addressOffset** (mandatory; type: ***unsignedLongintExpression*** (see C.3.8)) describes, in addressing units from the containing **addressSpace/addressUnitBits** element, the offset from the start of the **addressSpace**.

d) **range** (mandatory; type: ***unsignedPositiveLongintExpression*** (see C.3.10)) gives the address range of an address space segment. This is expressed as the number of addressable units of the address space segment. The size of an addressable unit is defined inside the **addressUnitBits** element.

e) **vendorExtensions** (optional) holds any vendor-specific data from other namespaces, which is applicable to this address space. See C.24.

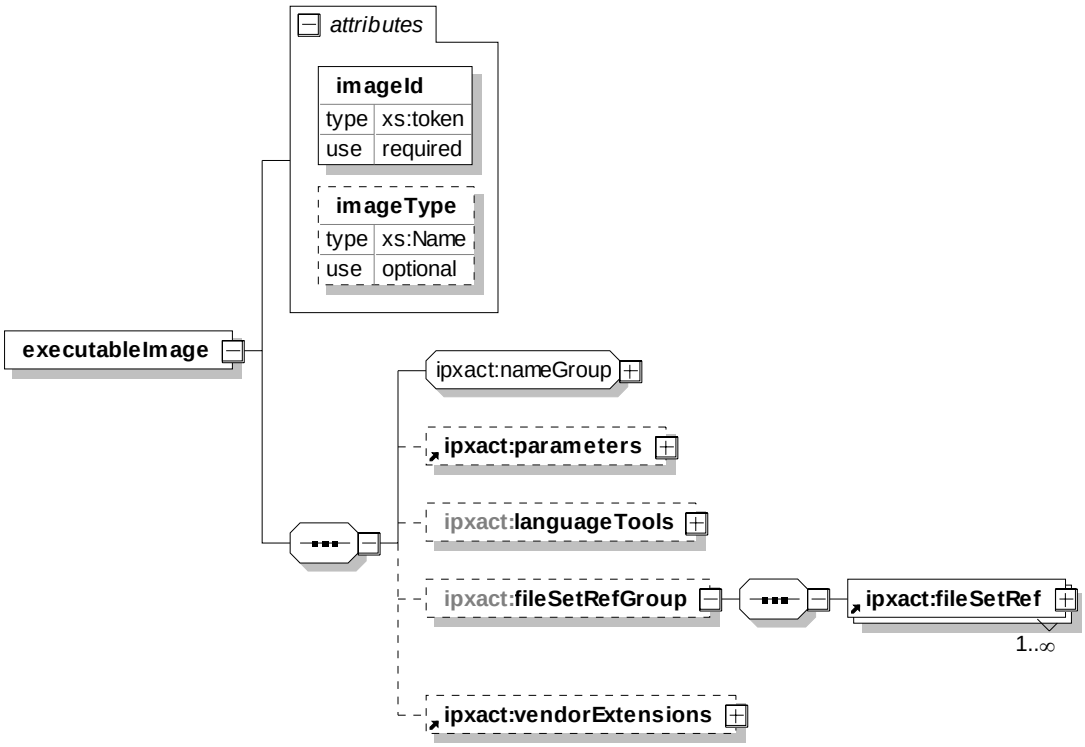
See also: SCR 9.7.

6.8.3 executableImage

6.8.3.1 Schema

The following schema details the information contained in the **executableImage** element, which may appear inside an **addressSpace** element.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



6.8.3.2 Description

The **executableImage** element describes the details of an executable image that can be loaded and executed in this address space on the processor to which this master bus interface belongs and contains the following attributes and elements.

a) **imageId** (mandatory; type ***token***) attribute uniquely identifies the **executableImage** for reference in a **fileSet/function/fileRef**.

b) **imageType** (optional; type ***Name***) attribute can describe the binary executable format (e.g., raw binary). The list of possible values is user-defined.

c) ***nameGroup*** group is defined in C.12. The **name** element shall be unique within the containing **segments** element.

d) **parameters** (optional) specifies any parameter data value(s) for this executable object. See C.18.

e) **languageTools** (optional) contains further elements to describe the information needed to build the executable image. See 6.8.4.

f) **fileSetRefGroup** (optional) element contains a list of **fileSetRef** subelements, each one containing the name of a file set associated with this **executableImage**. See 6.15.

g) **vendorExtensions** (optional) holds any vendor-specific data from other namespaces, which is applicable to this address space. See C.24.

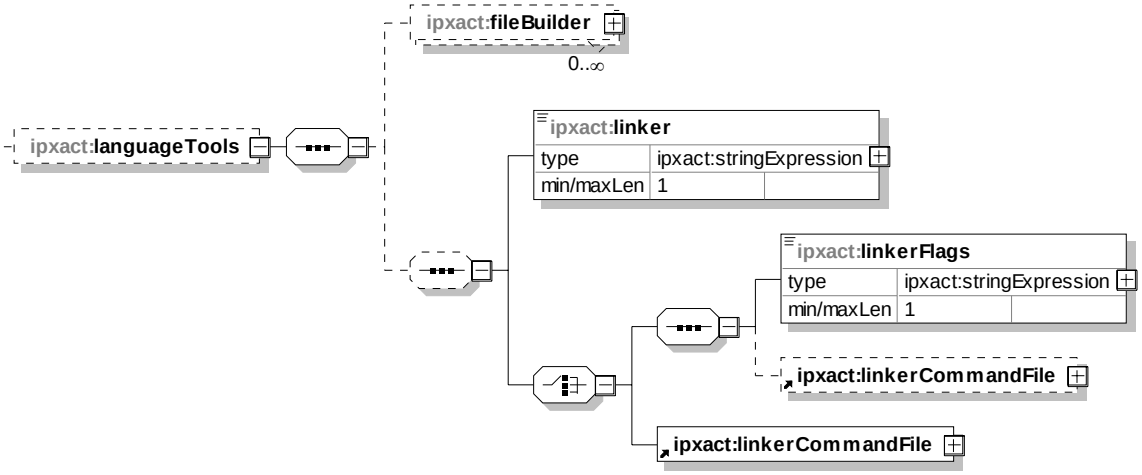
See also: SCR 9.3.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

6.8.4 languageTools

6.8.4.1 Schema

The following schema details the information contained in the **languageTools** element, which may appear as an element inside the **executableImage** element.



6.8.4.2 Description

The **languageTools** element contains the following list of optional elements to document a set of software tools used to create an executable binary documented by the parent **executableImage** element. Multiple **languageTools** information can be created to reflect various software tool sets that can create this executable binary file.

a) **fileBuilder** (optional) contains the information details of a compiler or assembler for software source code. See 6.8.5.

b) **linker** (optional; type: ***stringExpression*** (see C.3.3)) documents the link editor associated with the software tools described in **fileBuilder**.

c) A **languageTools** also contains one of the following:

a **linkerFlags** element and optionally a **linkerCommandFile**;
or a **linkerCommandFile**.

1) **linkerFlags** (optional; type: ***stringExpression*** (see C.3.3)) can also be associated with any **linker** information.

2) **linkerCommandFile** (optional) documents a file containing commands the linker follows.

The **linkerCommandFile** element contains information related to contents of the **linker** and **linkerFlags** elements, specifically about a file containing linker commands. It contains the following subelements.

i) **name** (mandatory; type: ***stringURIExpression*** (see C.3.4) documents the location and name of the file containing commands for the linker.

ii) **commandLineSwitch** (mandatory type: ***stringExpression*** (see C.3.3)) documents the flag on the command line specifying the linker command file.

iii) **enable** (mandatory; type: ***unsignedBitExpression*** (see C.3.5)) indicates whether to use this linker command file in the default scenario. The following also apply. For:

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

enable is **true** and a **generatorRef**, run the generator to link the **executableImage**; it may use the other elements to link the **executableImage**.

enable is **true** and no **generatorRef**, run the linker with the **-commandLineSwitch** *name* (the command file).

enable is **false**, run the linker with **linkerFlags**.

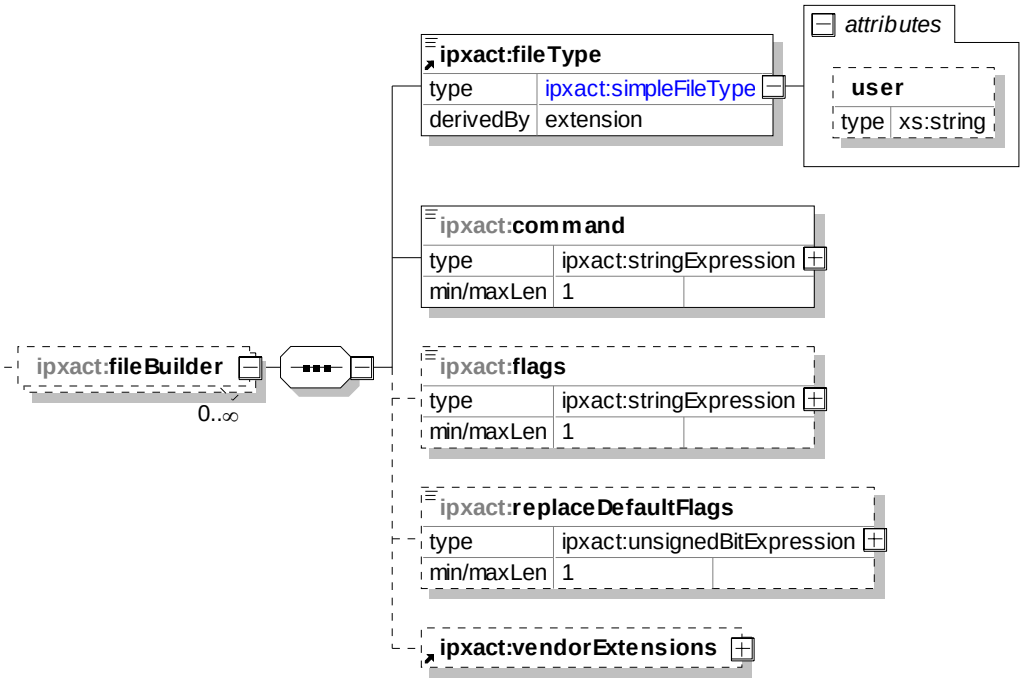
iv) **generatorRef** (optional; type: ***string***) references the generator (in the containing component) that creates and launches the linker command. There may be any number of these elements present. See 6.13.

v) **vendorExtensions** (optional) holds any vendor-specific data from other namespaces applicable to using this linker. See C.24.

6.8.5 fileBuilder

6.8.5.1 Schema

The following schema details the information contained in the **fileBuilder** element, which may appear as an element inside a **languageTools** element within the **executableImage** element.



6.8.5.2 Description

The **fileBuilder** element contains the following elements.

a) **fileType** (mandatory) indicates the type of file referenced by IP-XACT. If the required file type is not available, set the **fileType** to **user** and then set the **user** attribute to the desired file type. Any string value is valid in the **user** attribute, but the file may not be able to be processed as that specific type by a DE.

b) **command** (mandatory; type: ***stringExpression*** (***s***ee C.3.3)) defines a compiler or assembler tool that processes the software of this type.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

c) **flags** (optional; type: ***stringExpression*** (***s***ee C.3.3)) documents any flags to be passed along with the software tool command.

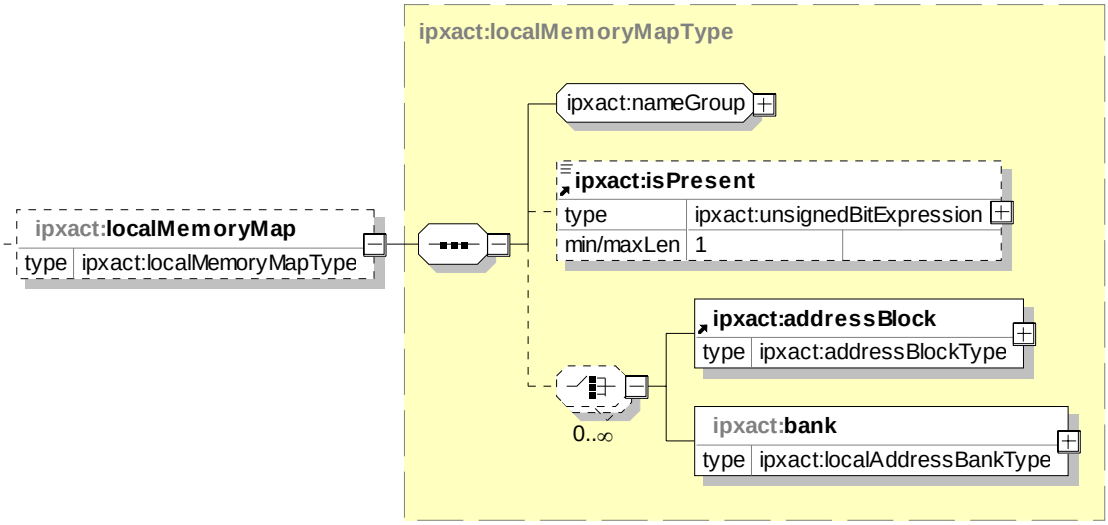
d) **replaceDefaultFlags** (optional; type: ***unsignedBitExpression*** (***s***ee C.3.5)) documents, when **true**, flags that replace any of the default flags from a build script generator. If **false**, the flags contained in the flags element are appended to the current command. If the value is **true** and the **flags** element is empty or does not exist, this has the effect of clearing all the flags in build script generator.

e) **vendorExtensions** (optional) holds vendor-specific data from other namespaces applicable to building this software source code file into an executable object file. See C.24.

6.8.6 Local memory map

6.8.6.1 Schema

The following schema details the information contained in the **localMemoryMap** element, which may appear inside an **addressSpace** element.



6.8.6.2 Description

Some processor components require specifying a memory map that is local to the component. *Local memory maps* (the **localMemoryMap** element in the **addressSpace** element of the component) are blocks of memory within a component that can only be accessed by the master interfaces of that component. If the master interface containing a local memory map is bridged from a slave interface (see 6.4.2), the local memory map is visible from this slave interface. **localMemoryMap** contains the following elements.

a) ***nameGroup*** group is describe in C.12.

b) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

c) **localMemoryMap** is any number of the following.

1) **addressBlock** describes a single block. See 6.9.2.

2) **bank** represents a collection of address blocks, banks, or subspace maps. See 6.9.5.

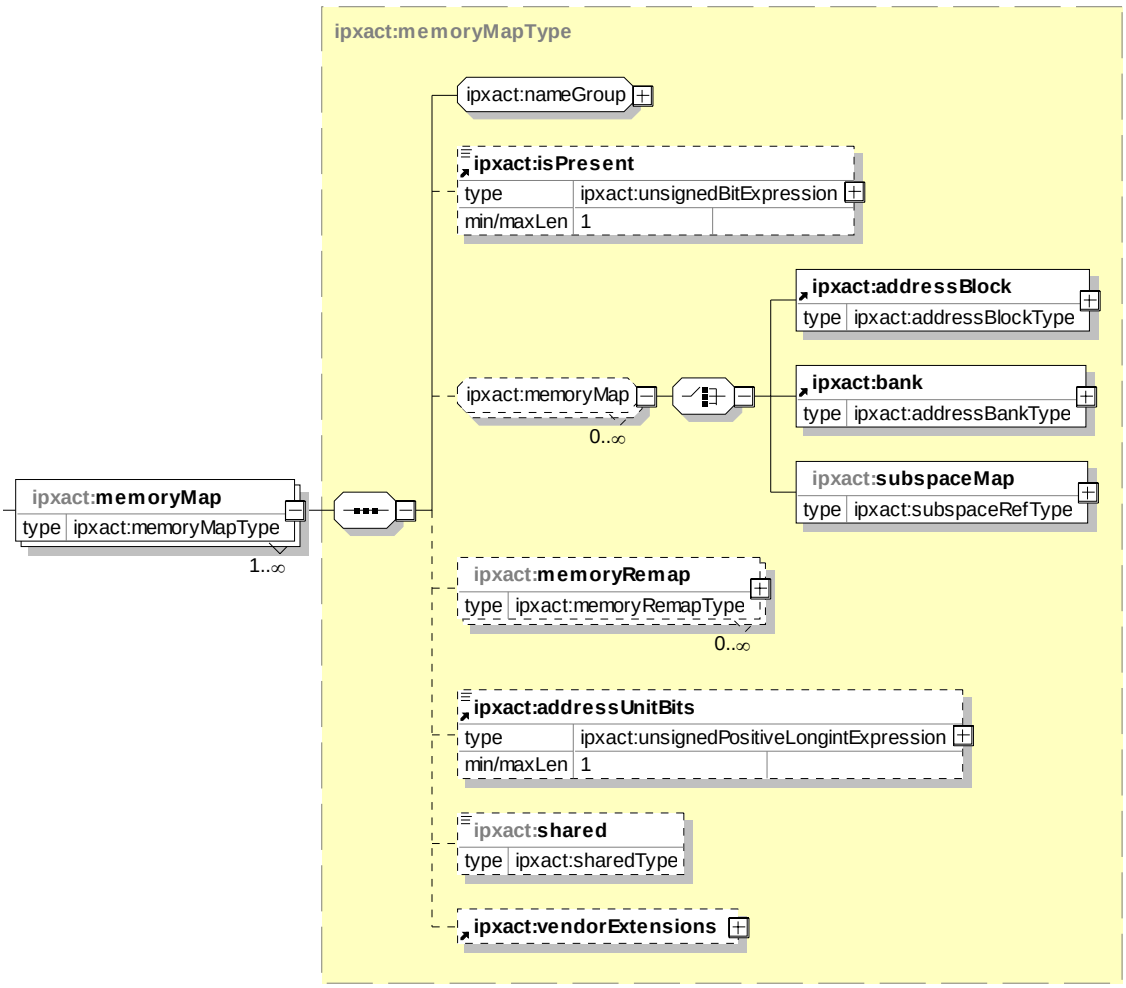
Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

6.9 Memory maps

6.9.1 memoryMaps

6.9.1.1 Schema

The following schema details the information contained in the **memoryMaps** element, which may appear as an element inside the **component** element.



6.9.1.2 Description

A memory map can be defined for each slave interface of a component. The **memoryMaps** element contains an unbounded list of **memoryMap** elements. The **memoryMap** elements are referenced by the component's slave interface. The **memoryMap** element contains an **id** (optional) attribute that assigns a unique identifier to the containing element for reference throughout the containing description. **memoryMap** contains the following mandatory and optional elements.

a) ***nameGroup*** group is defined in C.12. The **name** element shall be unique within the containing **memoryMaps** element.

b) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

c) ***memoryMap*** group (optional) is any number of the following.

1) **addressBlock** describes a single block. See 6.9.2.

2) **bank** represents a collections of address blocks, banks, or subspace maps. See 6.9.5.

3) **subspaceMap** maps the address subspaces of master interfaces into the slave's memory map. See 6.9.9.

d) The optional **memoryRemap** element describes additional address blocks, banks, and subspace maps of a slave bus interface in a specific remap state.

e) The optional **addressUnitBits** element (type: ***unsignedPositiveLongintExpression*** (see C.3.10)) defines the number of data bits in each address increment of the memory map. This is required to allow the elements in the memory map to define items such as register offsets..

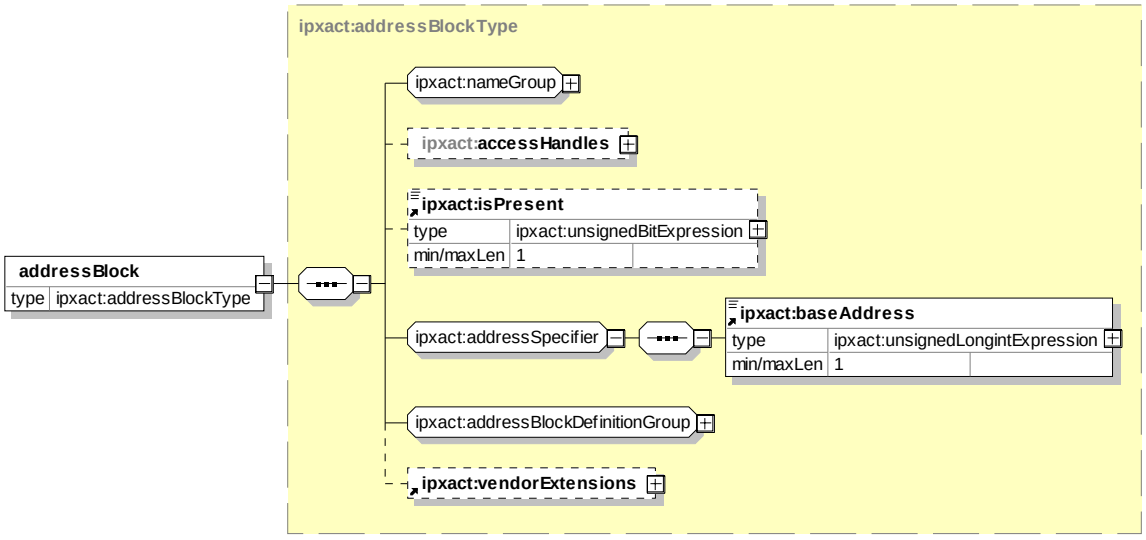
f) The optional **shared** element (default: **undefined**) defines the sharing properties of the memory map. When the value is **yes**, the contents of the **memoryMap** are shared by all the references to this **memoryMap**, when the value is **no**, the contents of the **memoryMap** are not shared and when the value is **undefined**, the sharing of the **memoryMap** is undefined.

g) **vendorExtensions** (optional) adds any extra vendor-specific data related to the memory map. See C.24.

6.9.2 Address block

6.9.2.1 Schema

The following schema details the information contained in the **addressBlock** element, which may appear in a **memoryMap** element. It is of type ***addressBlockType***.



6.9.2.2 Description

The **addressBlock** element describes a single, contiguous block of memory that is part of a memory map. **addressBlock** contains the following mandatory and optional elements.

a) ***nameGroup*** group is defined in C.12. The **name** element shall be unique within the containing **addressBlock** element.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

b) **accessHandles** (optional) specifies view-dependent naming for this address block within the corresponding RTL. See C.1.

c) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

d) ***addressSpecifier*** group includes the following.

baseAddress (mandatory; type ***unsignedLongintExpression*** (see C.3.8)) specifies the starting address of the block. The **baseAddress** is expressed in addressing units from the containing **memoryMap/addressUnitBits** or **localMemoryMap/addressUnitBits** element.

e) ***addressBlockDefinitionGroup*** group contains definition information about address blocks. See 6.9.3.

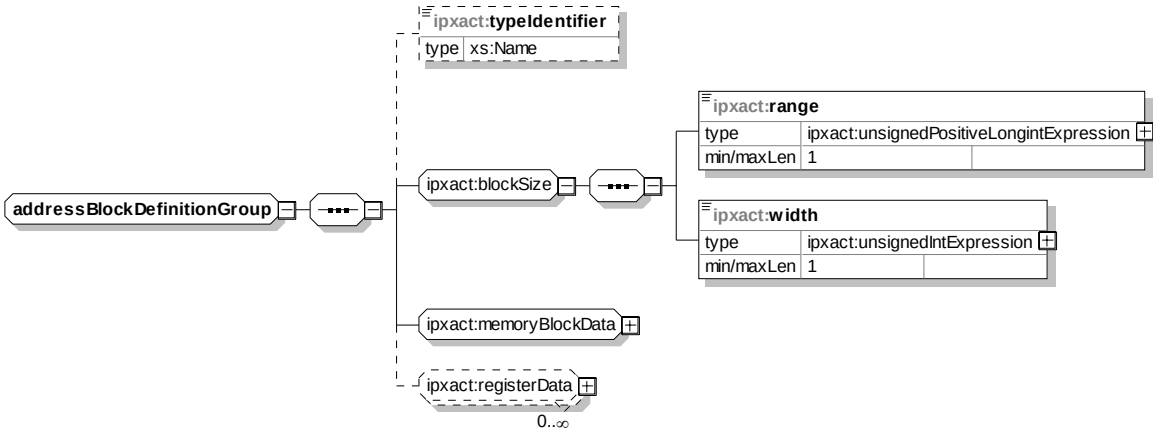
f) **vendorExtensions** (optional) adds any extra vendor-specific data related to the address block. See C.24.

See also: SCR 8.1 and SCR 8.16.

6.9.3 Address block definition group

6.9.3.1 Schema

The following schema details the information contained in the ***addressBlockDefinitionGroup*** group, which may appear in an **addressBlock** element.



6.9.3.2 Description

The ***addressBlockDefinitionGroup*** group describes the definition information about address blocks. It contains the following mandatory and optional elements.

a) **typeIdentifier** (optional; type: ***Name***) indicates multiple address block elements with the same **typeIdentifier** in the same description contain the exact same information for the elements in the **addressBlockDefinitionGroup**.

b) ***blockSize*** group includes the following.

1) **range** (mandatory; type: ***unsignedPositiveLongintExpression*** (see C.3.10)) gives the address range of an address block. This is expressed as the number of addressable units. The size of an addressable unit is defined inside the containing **memoryMap/addressUnitBits** or **memoryMap/addressUnitBits** element.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

2) **width** (mandatory; type: ***unsignedIntExpression*** (see C.3.7)) is the bit width of a row in the address block. A row in an address block sets the maximum single transfer size into the memory map allowed by the referencing bus interface and also defines the maximum size that a single register can be defined across an interconnection.

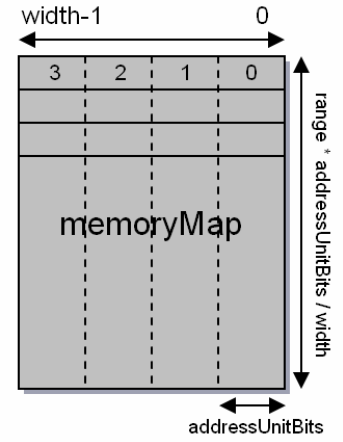
c) ***memoryBlockData*** group contains information about usage, access, volatility, and other parameters. See 6.9.4.

d) ***registerData*** group contains information about the grouping of bits into registers and fields. See 6.11.1.

The **range** and **width** elements are related by the following formulas.

number_of_bits_in_block = `addressUnitBits` × `range`

number_of_rows_in_block = *number_of_bits_in_block* / `width`



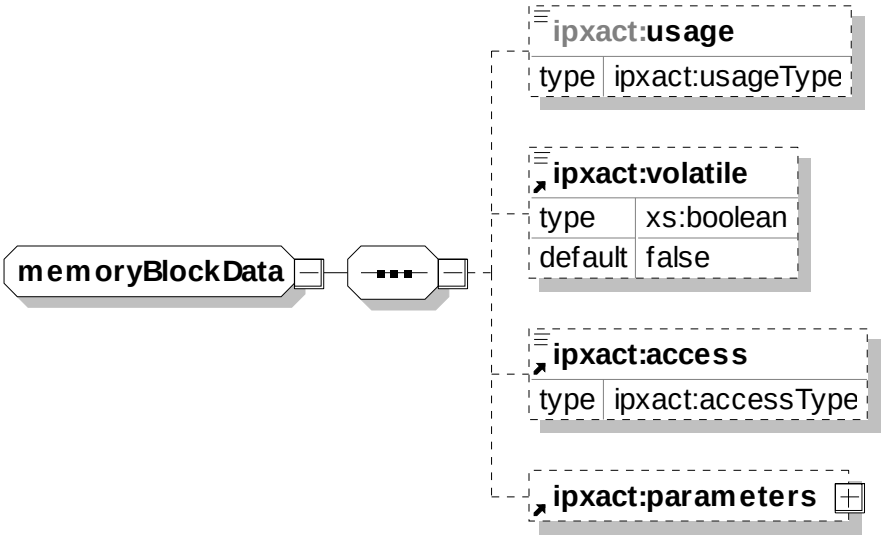
See also: SCR 8.1 and SCR 7.15.

6.9.4 memoryBlockData group

6.9.4.1 Schema

The following schema details the information contained in the ***memoryBlockData*** group, an optional part of both **addressBlock** and **bank**.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



6.9.4.2 Description

The ***memoryBlockData*** group is a collection of elements that contains further specification of **addressBlock** or **bank** elements. It contains the following elements.

a) **usage** (optional) specifies the type of usage for the address block or bank to which it belongs.

   1) For an **addressBlock**:

      i) **memory** defines, when the **access** element is set to **read-only**, the entire range of the **addressBlock** as a ROM. If the **access** element is set to **read-write**, the entire range of the **addressBlock** is a RAM. If the **access** element is set to **write-only**, the entire range of the **addressBlock** is a write-only memory. This usage type can contain virtual registers.

      ii) **register** defines the entire range of the **addressBlock** as possible locations for registers.

      iii) **reserved** defines the entire range of the **addressBlock** as reserved or for unknown usage to IP-XACT. This type shall not contain registers.

      iv) If unspecified, the presumed value for **usage** shall be **register** if the **addressBlock** contains **register** elements; otherwise it is **reserved**.

   2) For a **bank**:

      i) **memory** defines all containing **addressBlock** elements are of this access type.

      ii) **register** defines all containing **addressBlock** elements are of this access type.

      iii) **reserved** defines all containing **addressBlock** elements are of this access type.

      iv) Unspecified usage means the bank may contain a mixture of **memory**, **register**, and **reserved addressBlock** elements.

b) **volatile** (optional; type: ***boolean***; default: **false**) when **true** indicates the case of a write followed by read, or in the case of two consecutive reads, there is no guarantee what is returned by the read on the second transaction or that this return value is consistent with the write or read of the first transaction. This element implies there is some additional mechanism by which these registers can acquire new values other than reads/writes/resets and other access methods known to IP-XACT. If this element is not present, it is presumed to be **false** for a **field** and unspecified for a **bank**, **addressBlock**, or **register**.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

c) **access** (optional) specifies the accessibility of the data in the address block. If the **usage** element is **reserved**, this element has no meaning. If the **access** is not specified, the value shall be inherited from the containing **bank** or default to **read-write** if this element is contained in a **memoryMap**.

   i) **read-write** defines, when the **usage** element is **memory**, the entire range is a RAM. If the **usage** element is **register**, then any access type for a register or alternate register is allowed.

   ii) **read-only** defines, when the **usage** element is **memory**, the entire range is a ROM. If the **usage** element is **register**, then an access type shall be **read-only** for a register or alternate register.

   iii) **write-only** defines, when the **usage** element is **memory**, the entire range is a write-only memory. If the **usage** element is **register**, then an access type shall be **write-only** or **writeOnce** for a register or alternate register.

   iv) **read-writeOnce** defines, when the **usage** element is **memory**, the entire range is a RAM that is writable once after power up. If the **usage** element is **register**, then the access type for a register or alternate register shall be **read-only**, **read-writeOnce**, **write-only**, or **writeOnce**.

   v) **writeOnce** defines, when the **usage** element is **memory**, the entire range is a write-only memory that is writable once after power up. If the **usage** element is **register**, then the access type for a register or alternate register shall be **writeOnce**.

d) **parameters** (optional) details any additional parameters that describe the address block for generator usage. See C.18.

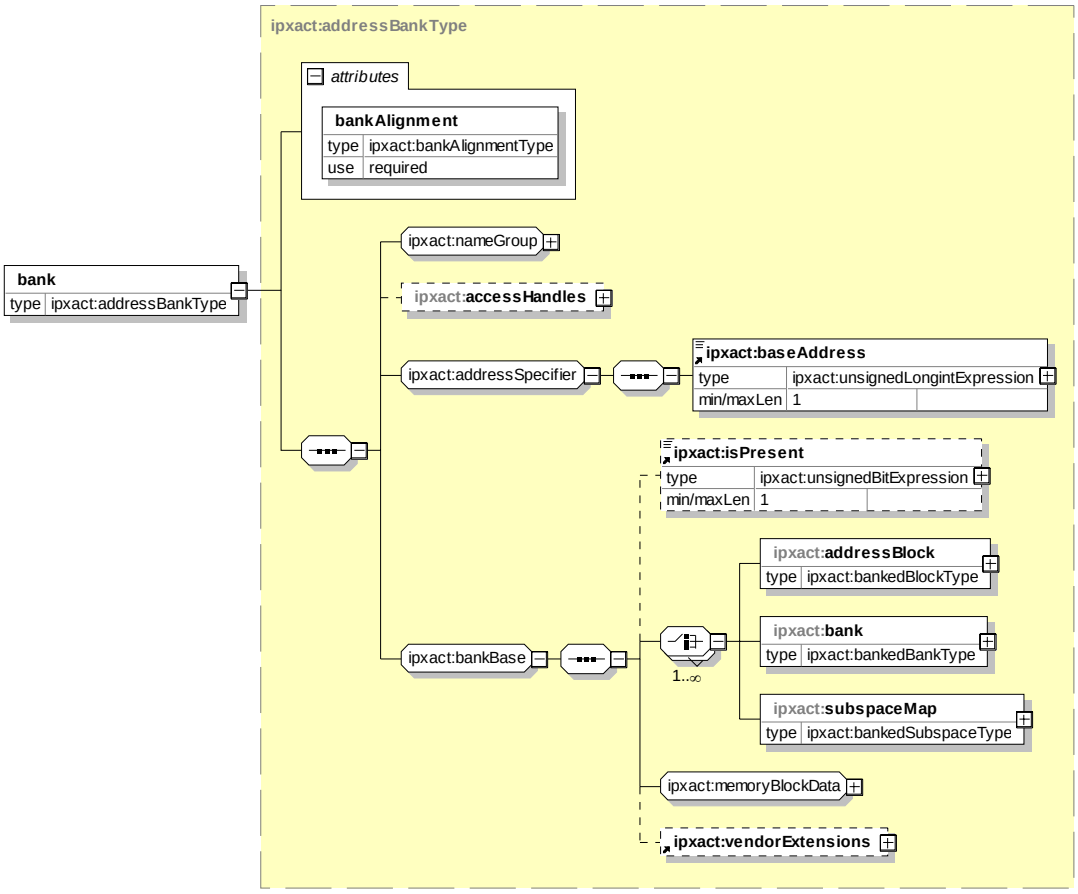
See also: SCR 8.3, SCR 8.4, SCR 8.6, SCR 8.7, SCR 8.9, SCR 8.10, SCR 8.11, SCR 8.13, SCR 8.14, and SCR 8.17.

6.9.5 Bank

6.9.5.1 Schema

The following schema details the information contained in the **bank** element, which can appear in a **memoryMap** element. It is of type ***addressBankType***.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



6.9.5.2 Description

The **bank** element allows multiple **addressBlocks**, **banks**, or **subspaceMaps** to be concatenated together horizontally or vertically as a single entity. It contains the following attributes and elements.

a) **bankAlignment** (mandatory) attribute organizes the bank:

   1) **parallel** specifies each item is located at the same base address with different bit offsets. The bit offset of the first item in the bank always starts at 0, the offset of the next items in the bank is equal to the widths of all the previous items.

   2) **serial** specifies the first item is located at the bank's base address. Each subsequent item is located at the previous item's address, plus the range of that item (adjusted for LAU and bus width considerations, rounded up to the next whole multiple). This allows the user to specify only a single base address for the bank and have each item assigned an address in sequence.

b) ***nameGroup*** group is defined in C.12. The **name** element shall be unique within the containing **bank** element.

c) **accessHandles** (optional) specifies view-dependent naming for this bank within the corresponding view. See C.1.

d) ***addressSpecifier*** group includes the following.

   baseAddress (mandatory; type: ***unsignedPositiveLongintExpression*** (see C.3.10)) specifies the starting address of the block. The **baseAddress** is expressed in addressing units from the containing **memoryMap/addressUnitBits** or **localMemoryMap/addressUnitBits** element.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

e) ***bankBase*** group includes the following. This group is later used inside the ***bankedBaseType*** type to create recursion.

1) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

2) A ***bankBase*** group also contains at least one of the following:

i) **addressBlock** is an address block that makes up part of the bank. See 6.9.6.

ii) **bank** is a bank within the bank. This allows for complex configurations with nested banks. See 6.9.7.

iii) **subspaceMap** is a reference to the master's address map for inclusion in the bank. See 6.9.9.

3) ***memoryBlockData*** group contains information about usage, access, volatility, and other parameters. See 6.9.4.

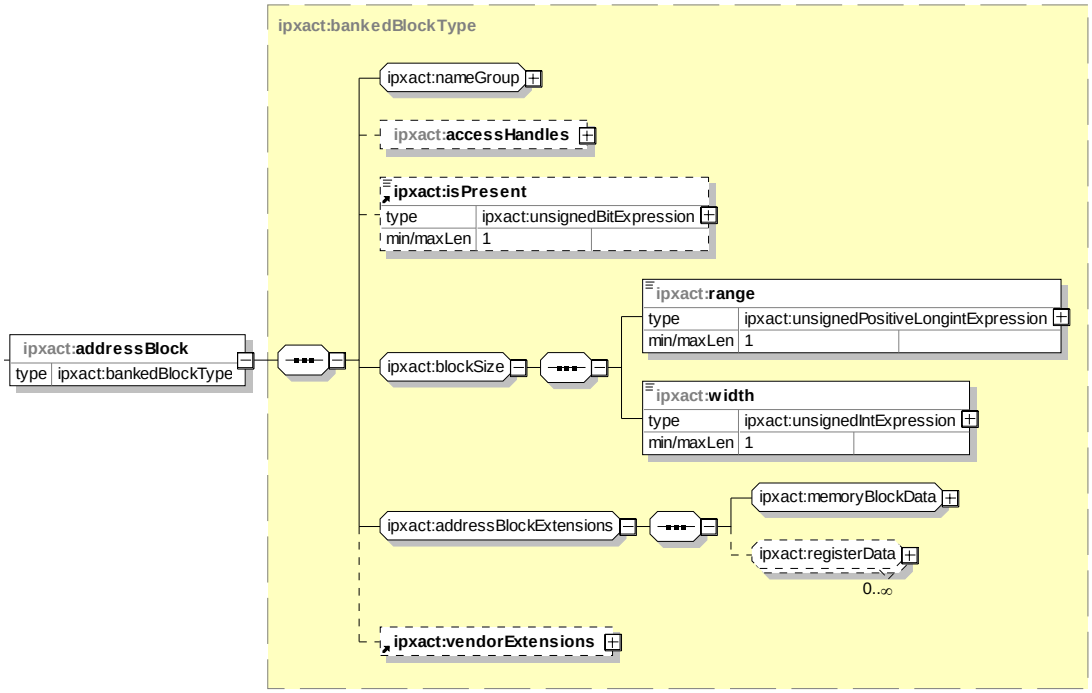
4) **vendorExtensions** adds any extra vendor-specific data related to this bank. See C.24.

See also: SCR 8.2.

6.9.6 Banked address block

6.9.6.1 Schema

The following schema details the information contained in the **addressBlock** element, which can appear in a **bank** element. It is of type ***bankedBlockType***.



6.9.6.2 Description

The **addressBlock** element inside a bank element describes a single, contiguous block of memory that is part of a bank. **addressBlock** contains the following mandatory and optional elements.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

a) ***nameGroup*** group is defined in C.12. The **name** element shall be unique within the containing **addressBlock** element.

b) **accessHandles** (optional) specifies view-dependent naming for this address block within the corresponding RTL. See C.1.

c) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

d) ***blockSize*** group includes the following.

   1) **range** (mandatory; type: ***unsignedPositiveLongintExpression*** (see C.3.10)) gives the address range of an address block. This is expressed as the number of addressable units of the memory map. The size of an addressable unit is defined inside the containing **memoryMap/addressUnitBits** or **localMemoryMap/addressUnitBits** element.

   2) **width** (mandatory; type: ***unsignedIntExpression*** (see C.3.7)) is the bit width of a row in the address block.

e) ***addressBlockExtensions*** group contains optional elements commonly added to various types of address blocks in a memory map, including the following.

   1) ***memoryBlockData*** group contains information about usage, access, volatility, and other parameters. See 6.9.4.

   2) ***registerData*** group contains information about the grouping of bits into registers and fields. See 6.11.1.

f) **vendorExtensions** (optional) adds any extra vendor-specific data related to the address block. See C.24.

NOTE—The ***bankedBlockType*** of an **addressBlock** element is almost identical to the ***addressBlockType*** of an **addressBlock** element (see 6.9.2); the only difference is there is no **baseAddress** or **typeIdentifier** in the ***bankedBlockType*** version.

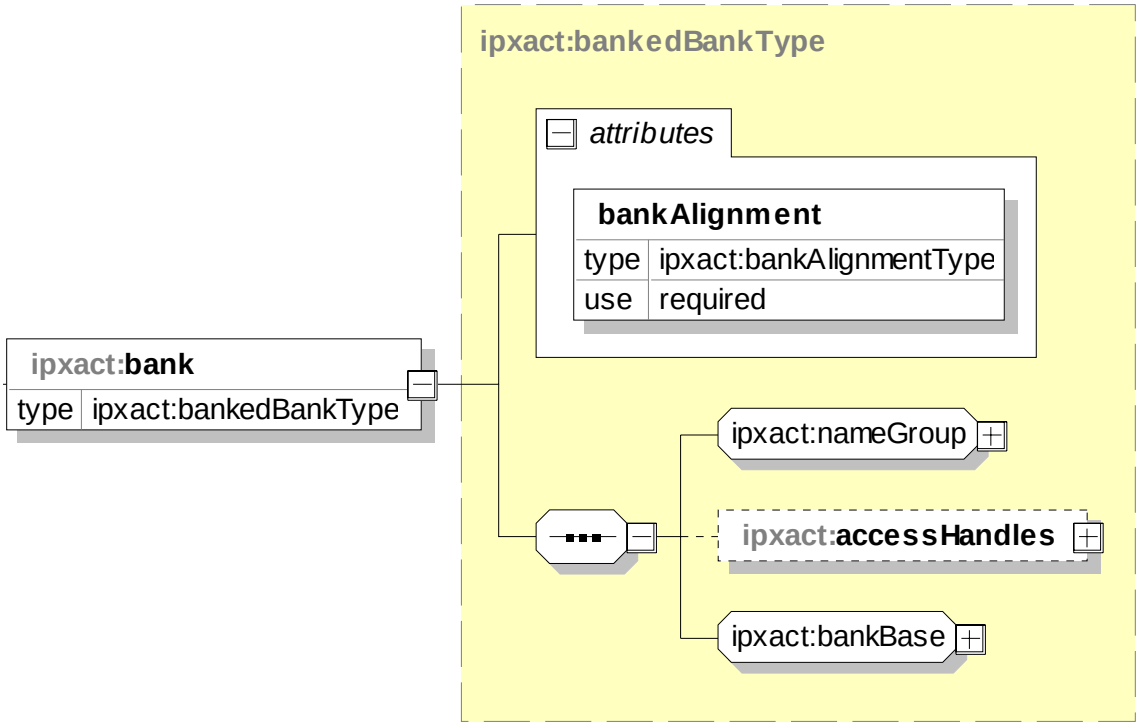
See also: SCR 7.5.

6.9.7 Banked bank

6.9.7.1 Schema

The following schema details the information contained in the nested **bank** element, which can appear in another **bank** element. It is of type ***bankBankType***.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



6.9.7.2 Description

The **bank** element within another bank element (banked bank) allows multiple address **blocks**, **banks**, or **subspaceMaps** to be to be concatenated together horizontally or vertically as a single entity. It contains the following attributes and elements.

a) **bankAlignment** (mandatory) attribute organizes the bank:

   1) **parallel** specifies each item is located at the same base address with different bit offsets. The bit offset of the first item in the bank always starts at 0, the offset of the next items in the bank is equal to the widths of all the previous items.

   2) **serial** specifies the first item is located at the bank's base address. Each subsequent item is located at the previous item's address, plus the range of that item (adjusted for LAU and bus width considerations, rounded up to the next whole multiple). This allows the user to specify only a single base address for the bank and have each item assigned an address in sequence.

b) ***nameGroup*** group is defined in C.12. The **name** element shall be unique within the containing **bank** element.

c) **accessHandles** (optional) specifies view-dependent naming for this bank within the corresponding RTL. See C.1.

d) The **bank** element of type ***bankedBankType*** contains the ***bankBase*** group. This group is defined inside the **bank** element of type ***addressBankType***. See 6.9.5. The effect of its inclusion here creates recursion, whereby banks maybe included inside banks included inside banks.

NOTE—A banked bank is similar to a bank in a memory map (see 6.9.5); the only difference is there is no **baseAddress** element in a **bank** of type ***bankedBankType***.

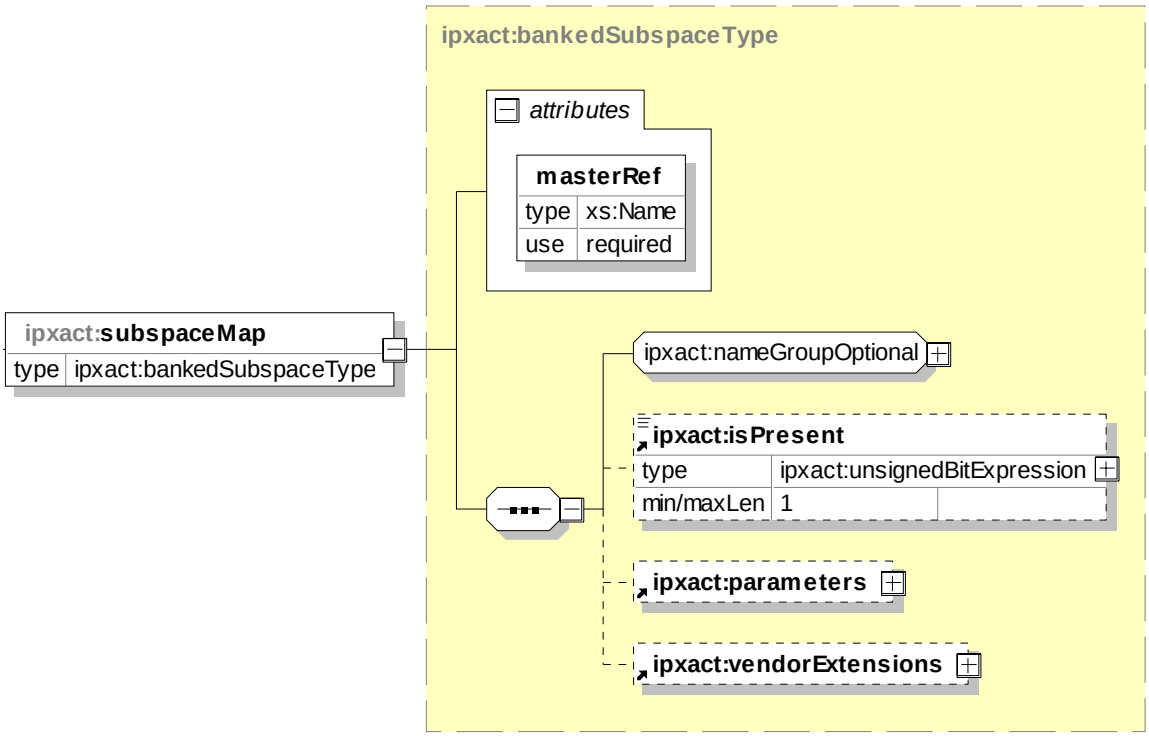
See also: SCR 8.2.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

6.9.8 Banked subspace

6.9.8.1 Schema

The following schema details the information contained in the **subspaceMap** element, which can appear in a **bank** element. It is of type ***bankSubspaceType***.



6.9.8.2 Description

The **subspaceMap** element, within an address bank, allows a bank to map the address space of a master interface into the bank. It contains the following elements.

a) **masterRef** attribute (mandatory; type: ***Name***) contains the name of the master interface whose address space needs to be mapped. This shall reference a bus interface name with an interface mode of master (see 6.5.3).

b) ***nameGroupOptional*** group is defined in C.13. The **name** of the **addressBlock**, **subspaceMap**, and **bank** shall be unique within the containing **bank** element.

c) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

d) **parameters** details any additional parameters that apply to the **subspaceMap**. See C.18.

e) **vendorExtensions** adds any extra vendor-specific data related to the **subspaceMap**. See C.24.

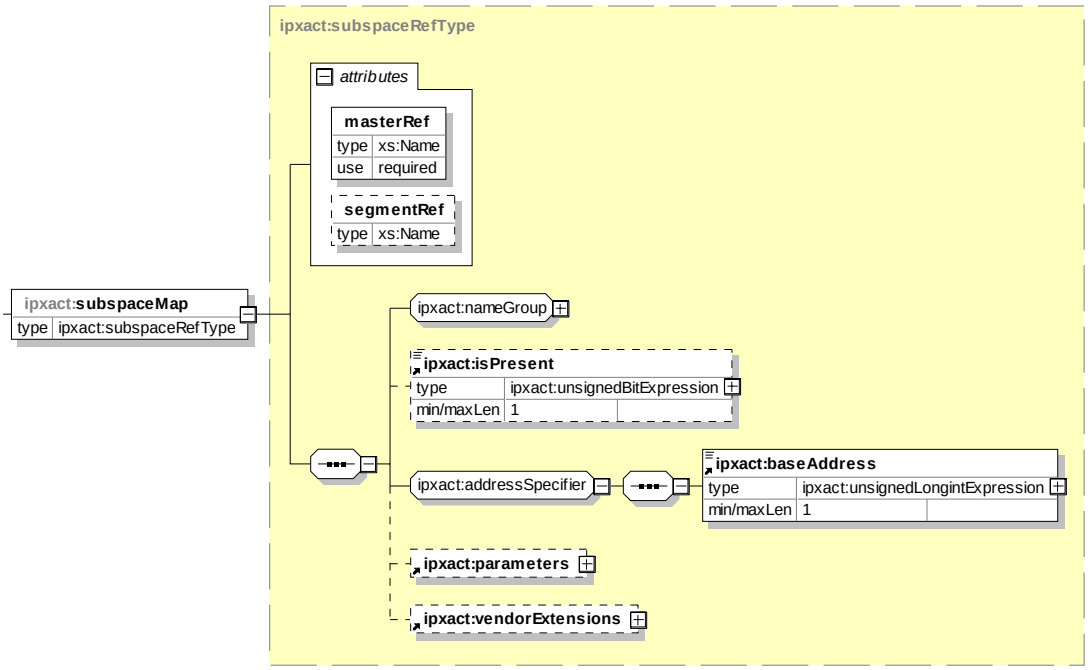
See also: SCR 8.2.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

6.9.9 Subspace map

6.9.9.1 Schema

The following schema details the information contained in the **subspaceMap** element, which can appear in a **memoryMap** element. It is of type ***subspaceRefType***.



6.9.9.2 Description

The **subspaceMap** element maps the address space of a master interface from an opaque bus bridge into the memory map. It contains the following elements and attributes.

a) **masterRef** (mandatory; type: ***Name***) attribute contains the name of the master interface whose address space needs to be mapped. This shall reference a bus interface name with an interface mode of master (see 6.5.3).

b) **segmentRef** (optional; type: ***Name***) references a **segment** in the **addressSpace** referred by the **masterRef** attribute. If the **segmentRef** attribute is not present, the entire **addressSpace** is presumed to be referenced.

c) ***nameGroup*** group is defined in C.12. The **name** of the **addressBlock**, **subspaceMap**, **bank,** and **memoryRemap** shall be unique within the containing **memoryMap**, **localMemoryMap**, or **memoryRemap** element.

d) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

e) ***addressSpecifier*** group includes the following.

baseAddress (mandatory; type: ***unsignedLongintExpression*** (see C.3.8)) specifies the starting address of the block. The **baseAddress** is expressed in addressing units from the containing **memoryMap/addressUnitBits** or **localMemoryMap/addressUnitBits** element.

f) **parameters** (optional) details any additional parameters that apply to the **subspaceMap**. See C.18.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

g) **vendorExtensions** (optional) adds any extra vendor-specific data related to the **subspaceMap**. See C.24.

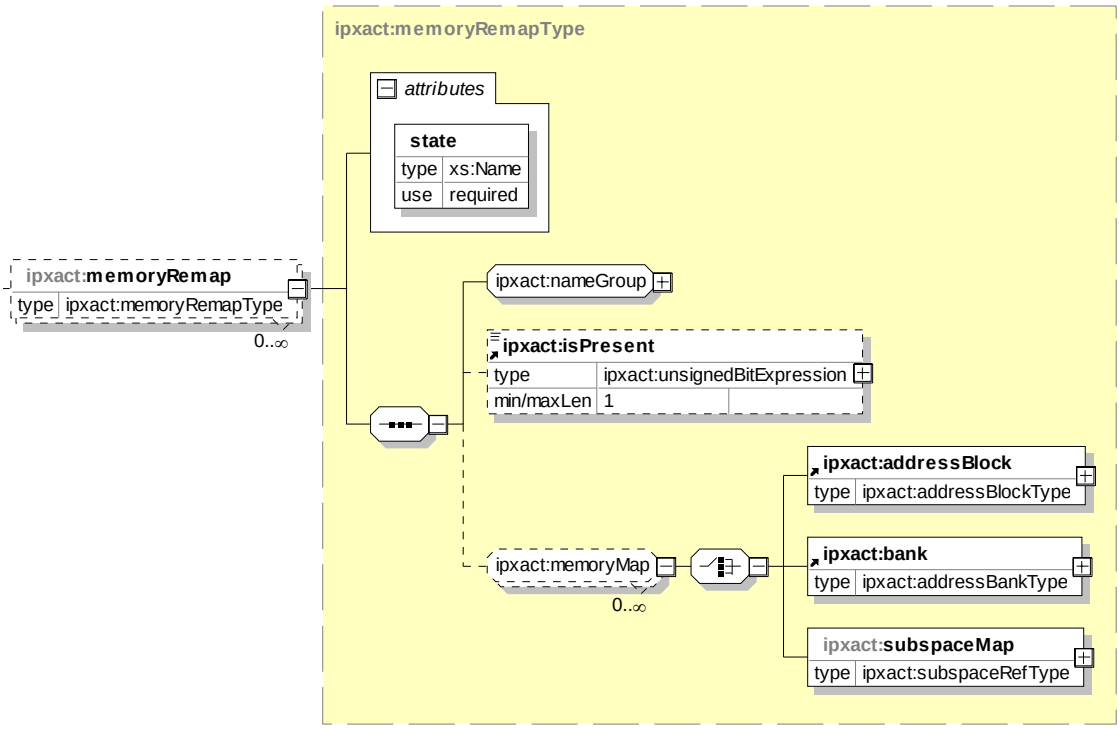
See also: SCR 3.17, SCR 9.8, SCR 15.1, and SCR 15.8.

6.10 Remapping

6.10.1 Memory remap

6.10.1.1 Schema

The following schema details the information contained in the **memoryRemap** element, which can appear in a **memoryMap** element. It is of type ***memoryRemapType***.



6.10.1.2 Description

The **memoryRemap** element describes additional **addressBlock**s, **bank**s, and **subspaceMap**s that are mapped on the referencing slave bus interface in a specific remap **state**. If multiple **memoryRemap/state** attributes are active, then the first **memoryRemap** listed shall be selected. This element contains the following elements, attributes, and groups.

a) **state** attribute (mandatory; type: ***Name***) identifies the remap state name for which the optional memory map elements are active. The **state** attribute shall reference a **remapState/name** in the containing description. The **state** attribute of all **memoryRemap** elements contained in a single **memoryMap** element shall be unique.

b) ***nameGroup*** group is defined in C.12. The **name** of the **addressBlock**, **subspaceMap**, **bank**, and **memoryRemap** shall be unique within the containing **memoryMap** element.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

c) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

d) ***memoryMap*** group (optional) is any number of the following.

1) **addressBlock** describes a single block. See 6.9.2.

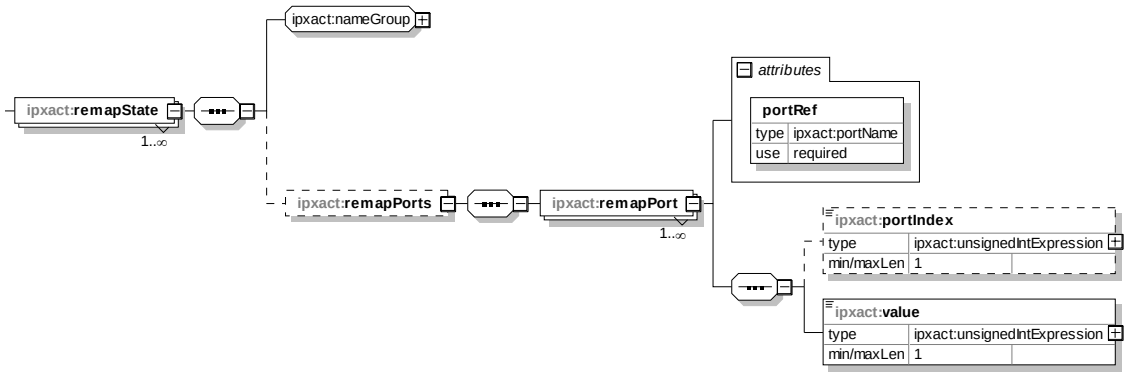
2) **bank** represents a collections of address blocks, banks, or subspace maps. See 6.9.5.

3) **subspaceMap** maps the address subspaces of master interfaces into the slave's memory map. See 6.9.9.

6.10.2 Remap states

6.10.2.1 Schema

The following schema details the information contained in the **remapStates** element, which may appear as an element inside a **component** element. This element may contain one or more **remapState** elements.



6.10.2.2 Description

A **remapStates** element describes a set of one or more **remapState** elements. Each **remapState** element defines a conditional remap state where each state is conditioned by a remap port specified with a **remapPort** element. A **remapState** element does not specify remapping addresses. The remapping addresses are defined by the **memoryRemap** element (of a **memoryMap** element) and its **state** attribute refers to the **remapState** element's name explained in this section.

remapState contains the following elements and attributes.

a) ***nameGroup*** group is defined in C.12. The **name** element shall be unique within the containing **remapStates** element.

b) **remapPorts** (optional) contains a list of **remapPort** elements. **remapPort** (mandatory) specifies when the remap state gets effective. A collection of **remapPort** elements make up the condition for this remap state. All elements shall be true for the remap state to be enabled. The type of this element is of ***scaledNonNegativeInteger***. This element contains the logical value of the single port bit specified by the following attributes.

1) **portRef** (mandatory; type: ***portName***) attribute is the name of the port in the containing description for which this logic value comparison is assigned. See 6.12.7.

2) **portIndex** (optional; type: ***unsignedIntExpression*** (see C.3.7)) attribute references the index of a port in the containing description, when the port being referenced is vectored.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

3) **value** (mandatory; type: ***unsignedIntExpression*** (see C.3.7)) is the value necessary so the specified port activates the **remapState**.

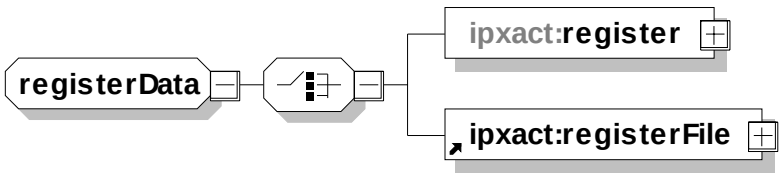
Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

6.11 Registers

6.11.1 Register data

6.11.1.1 Schema

The following schema details the information contained in the ***registerData*** group that may appear as an element inside the **addressBlock** element.



6.11.1.2 Description

The ***registerData*** group describes registers and register files. The containing **register/name** elements, the **register/alternateRegister/name** elements and the **registerFile/name** elements shall be unique within the containing **addressBlock** element. The ***registerData*** group contains one of the following elements.

a) **register** defines a list of registers contained in this **addressBlock**. See 6.11.2.

b) **registerFile** defines a list of register files contained in this **addressBlock**. See 6.11.6.

6.11.2 Register

6.11.2.1 Schema

The following schema details the information contained in the **register** element, which is contained in the ***registerData*** group that may appear as an element inside the **addressBlock** element. This element describes a register.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

6.11.2.2 Description

A **register** element describes a register in an address block or register file. The bits in the register are numbered from `size-1` down to `0`, with bit zero (`0`) being the least significant bit. **register** contains the following elements.

a) ***nameGroup*** group is defined in C.12. The **name** element shall be unique within the containing **register** element.

b) **accessHandles** (optional) specifies view-dependent naming for this register within the corresponding RTL. See C.1.

c) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

d) **dim** (optional; type: ***unsignedLongIntExpression*** (see ~~C.3.7~~C.3.8)) assigns an unbounded dimension to the register, so it is repeated as many times as the value of the **dim** elements. For multi-dimensional register arrays, the memory layout is presumed to follow the IEEE Std 1666™ [B4] (SystemC) language rules.

e) **addressOffset** (mandatory; type: ***unsignedLongintExpression*** (see C.3.8)) describes the offset from the start of the containing **addressBlock** or **registerFile** element. The **addressOffset** is expressed in addressing units from the containing **memoryMap/addressUnitBits** or **localMemoryMap/addressUnitBits** element.

f) ***registerDefinitionGroup*** group describes additional elements for a register. See 6.11.3

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

g) **alternateRegisters** (optional) describes alternate descriptions for the containing register. See 6.11.4

h) **parameters** (optional) describes any parameter names and types when the register width can be parameterized. See C.18.

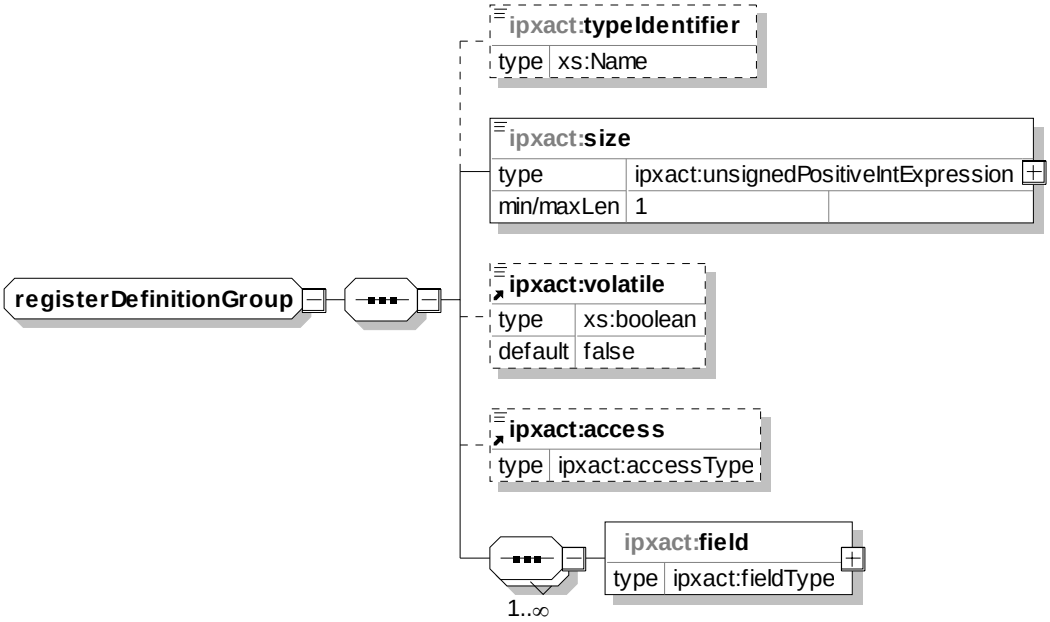
i) **vendorExtensions** (optional) adds any extra vendor-specific data related to this register. See C.24.

See also: SCR 7.1, SCR 7.2, SCR 7.3, SCR 7.4, SCR 7.5, SCR 7.7, SCR 7.8, SCR 7.9, SCR 7.13, SCR 8.3, SCR 8.4, SCR 8.5, SCR 8.7, SCR 8.8, and SCR 8.9.

6.11.3 Register definition group

6.11.3.1 Schema

The following schema details the information contained in the ***registerDefinitionGroup*** group, which is contained in the **register** element. This group describes register definition information.



6.11.3.2 Description

A ***registerDefinitionGroup*** group contains the following elements.

a) **typeIdentifier** (optional) indicates multiple register elements with the same **typeIdentifier** in the same description contain the exact same information for the elements in the **registerDefinitionGroup**.

b) **size** (mandatory; type: ***unsignedPositiveIntExpression*** (see C.3.10)) is the width of the register, counting in bits.

c) **volatile** (optional; type: ***boolean***; default: **false**) when **true** indicates in the case of a write followed by read, or in the case of two consecutive reads, there is no guarantee as to what is returned by the read on the second transaction or that this return value is consistent with the write or read of the first transaction. The element implies there is some additional mechanism by which this register can acquire new values other than by reads/writes/resets and other access methods known to IP-XACT. If this element is not present, no presumptions can be made about its value.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

d) **access** (optional) indicates the accessibility of the register. If this is not present, the **access** is inherited from the containing **addressBlock**. This element can take one of the following values.

   1) **read-write**: Both read and write transactions may have an effect on this register. Write transactions may affect the contents of the register and read transactions return a value related to the values in the register.

   2) **read-only**: A read transaction to this address returns a value related to the values in the register. A write transaction to this register has undefined results.

   3) **write-only**: A write transaction to this address affects the contents of the register. A read transaction to this register has undefined results.

   4) **read-writeOnce**: Both read and write transactions may have an effect on this register. Only the first write transaction, after an event that caused the reset value of the register to be loaded, may affect the contents of the register and read transactions return a value related to the values in the register.

   5) **writeOnce**: Only the first write transaction, after an event that caused the reset value of the register to be loaded, affects the contents of the register. A read transaction to this register has undefined results.

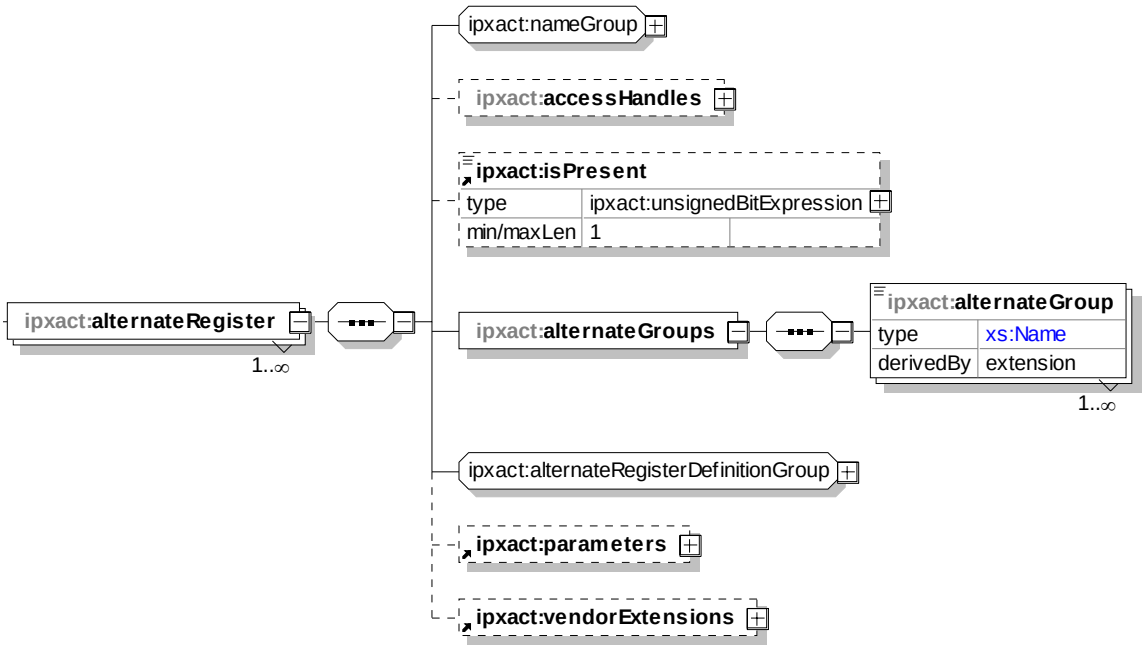
e) **field** (optional) describes a bit field within a register. See 6.11.8.

See also: SCR 7.1, SCR 7.2, SCR 7.3, SCR 7.4, SCR 7.5, SCR 7.7, SCR 7.8, SCR 7.9, SCR 7.13, SCR 8.3, SCR 8.4, SCR 8.5, SCR 8.7, SCR 8.8, SCR 8.9, SCR 8.11, SCR 8.12, SCR 8.14, and SCR 8.15.

6.11.4 Alternate registers

6.11.4.1 Schema

The following schema details the information contained in the **alternateRegister** element. Each **alternateRegister** element contained within the same **alternateRegisters** element provides an alternate description for the containing register element.



Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

6.11.4.2 Description

An **alternateRegister** element contains an alternate definition for the containing register. **alternateRegister** contains the following elements.

a) ***nameGroup*** group is defined in C.12. The **name** element shall be unique within the containing **alternateRegister** element.

b) **accessHandles** (optional) specifies view-dependent naming for this register within the corresponding RTL. See C.1.

c) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

d) **alternateGroups** (mandatory) defines an unbounded list of grouping names for which this alternate description belongs.

> **alternateGroup** (mandatory; type: ***Name***) defines a grouping name for this alternate register description. All **alternateGroup** elements shall be unique for each containing **register**.

e) ***alternateRegisterDefinitionGroup*** group describes additional elements for an alternate register. See 6.11.5

f) **parameters** (optional) describes any parameter names and types when the register width can be parameterized. See C.18.

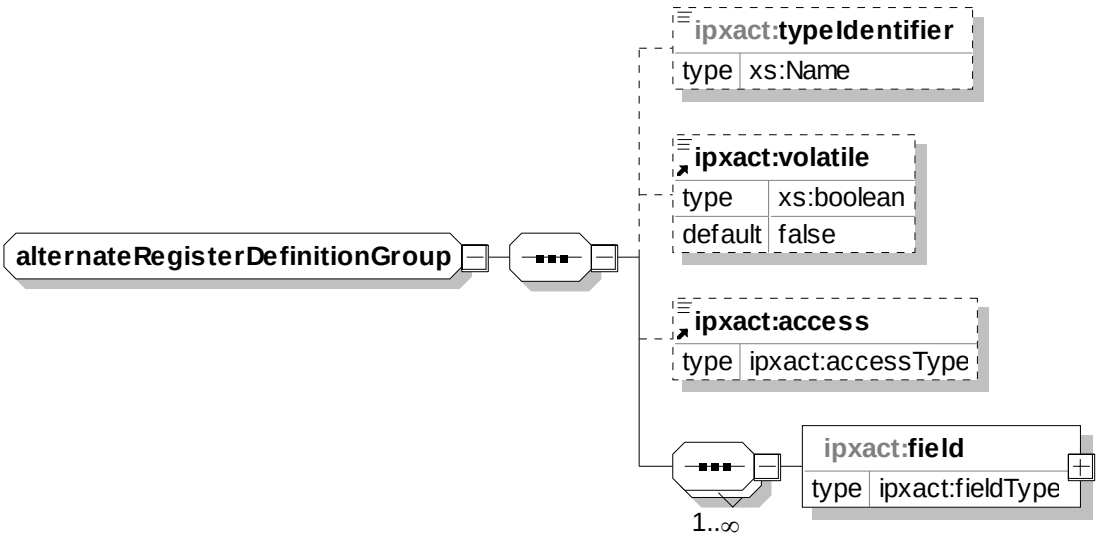
g) **vendorExtensions** (optional) adds any extra vendor-specific data related to this register. See C.24.

See also: SCR 7.1, SCR 7.2, SCR 7.3, SCR 7.4, SCR 7.7, SCR 7.8, SCR 7.9, SCR 7.13, SCR 8.3, SCR 8.4, SCR 8.5, SCR 8.7, SCR 8.8, SCR 8.9, SCR 8.11, SCR 8.12, SCR 8.14, SCR 8.15, and SCR 8.18.

6.11.5 Alternate register definition group

6.11.5.1 Schema

The following schema details the information contained in the ***alternateRegisterDefinitionGroup*** group, which is contained in the **alternateRegister** element. This group describes alternate register definition information.



Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

6.11.5.2 Description

A ***alternateRegisterDefinitionGroup*** group contains the following elements.

a) **typeIdentifier** (optional; type: ***Name***) indicates multiple register elements with the same **typeIdentifier** in the same description contain the exact same information for the elements in the ***alternateRegisterDefinitionGroup***.

b) **volatile** (optional; type: ***boolean***; default: **false**) when **true** indicates in the case of a write followed by read, or in the case of two consecutive reads, there is no guarantee as to what is returned by the read on the second transaction or that this return value is consistent with the write or read of the first transaction. The element implies there is some additional mechanism by which this register can acquire new values other than by reads/writes/resets and other access methods known to IP-XACT. If this element is not present, no presumptions can be made about its value.

c) **access** (optional) indicates the accessibility of the register. If this is not present, the **access** is inherited from the containing **addressBlock**. This element can take one of the following values.

   1) **read-write**: Both read and write transactions may have an effect on this register. Write transactions may affect the contents of the register and read transactions return a value related to the values in the register.

   2) **read-only**: A read transaction to this address returns a value related to the values in the register. A write transaction to this register has undefined results.

   3) **write-only**: A write transaction to this address affects the contents of the register. A read transaction to this register has undefined results.

   4) **read-writeOnce**: Both read and write transactions may have an effect on this register. Only the first write transaction, after power up, may affect the contents of the register and read transactions return a value related to the values in the register.

   5) **writeOnce**: Only the first write transaction, after power up, to this address affects the contents of the register. A read transaction to this register has undefined results.

d) **field** (optional) describes any bit fields in a register. See 6.11.8.

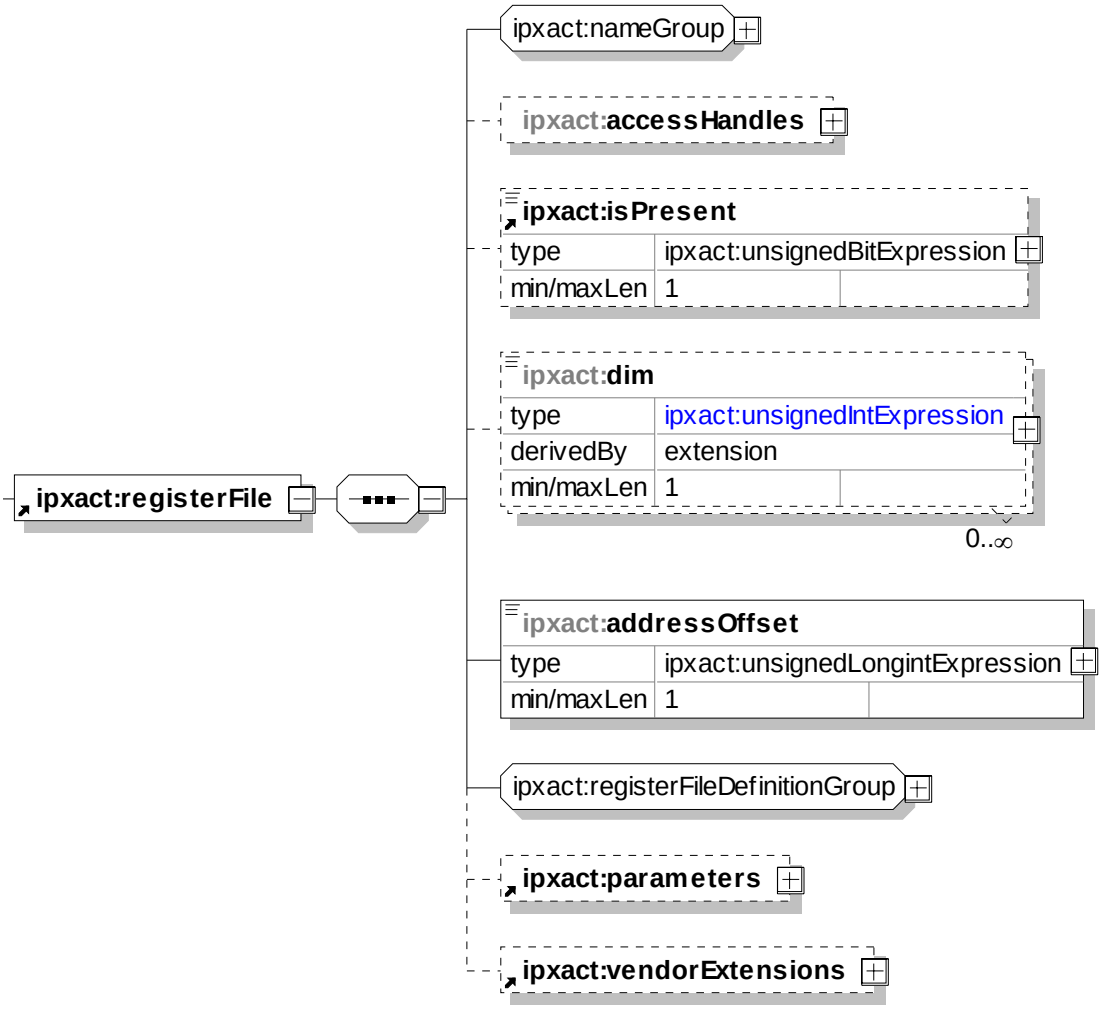
See also: SCR 7.1, SCR 7.2, SCR 7.3, SCR 7.4, SCR 7.7, SCR 7.8, SCR 7.9, SCR 7.13, SCR 8.3, SCR 8.4, SCR 8.5, SCR 8.7, SCR 8.8, and SCR 8.9.

6.11.6 Register file

6.11.6.1 Schema

The following schema details the information contained in the **registerFile** element, which is contained in the ***registerData*** group that may appear as an element inside the **addressBlock** element. This element describes a register file.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



6.11.6.2 Description

A **registerFile** element describes a grouping of registers in an address block or register file. **registerFile** contains the following elements.

a) ***nameGroup*** group is defined in C.12. The **name** element shall be unique within the containing **registerFile** element.

b) **accessHandles** (optional) specifies view-dependent naming for this register files within the corresponding RTL. See C.1.

c) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

d) **dim** (optional type: ***unsigned~~Long~~Int~~i~~ntExpression*** (see ~~C.3.7~~C.3.8)) assigns an unbounded dimension to the register, so it is repeated as many times as the value of the **dim** elements. For multi-dimensional register arrays, the memory layout is presumed to follow the IEEE Std 1666™ [B4] (SystemC) language rules.

e) **addressOffset** (mandatory; type: ***unsignedLongintExpression*** (see C.3.8)) describes the offset from the start of the containing **addressBlock** or **registerFile** element. The **addressOffset** is

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

expressed in addressing units from the containing **memoryMap/addressUnitBits** or **localMemoryMap/addressUnitBits** element.

f) ***registerFileDefinitionGroup*** group describes additional elements for a register file. See 6.11.7.

g) **parameters** (optional) describes any parameter names and types when the register width can be parameterized. See C.18.

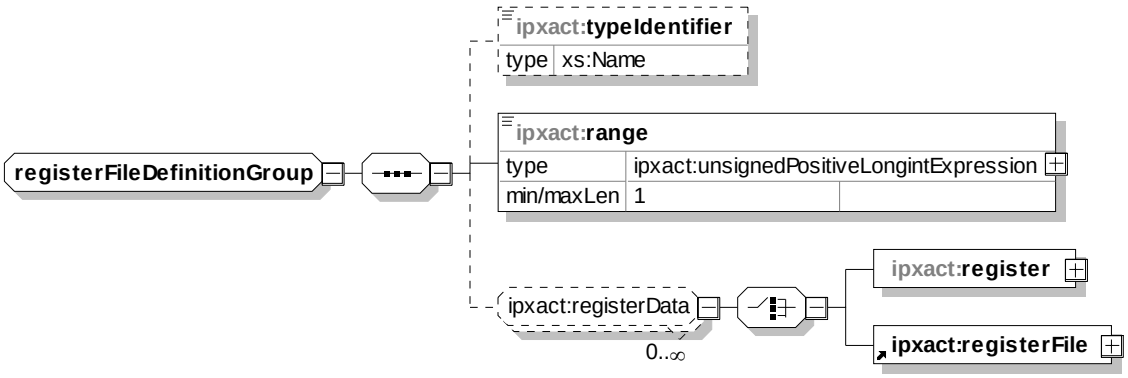
h) **vendorExtensions** (optional) adds any extra vendor-specific data related to this register. See C.24.

See also: SCR 7.6, SCR 7.7, and SCR 7.14.

6.11.7 Register file definition group

6.11.7.1 Schema

The following schema details the information contained in the ***registerFileDefinitionGroup*** group, which may appear as an element inside the **register** element. This element describes the reset value of the register.



6.11.7.2 Description

The ***registerFileDefinitionGroup*** group describes additional information for a register file.

a) **typeIdentifier** (optional; type: ***Name***) indicates multiple register elements with the same **typeIdentifier** in the same description contain the exact same information for the elements in the **registerDefinitionGroup**.

b) **range** (mandatory; type: ***unsignedPositiveLongintExpression*** (see C.3.10)) gives the range of a register file. This is expressed as the number of addressable units of the register file. The size of an addressable unit is defined inside the containing **memoryMap/addressUnitBits** element.

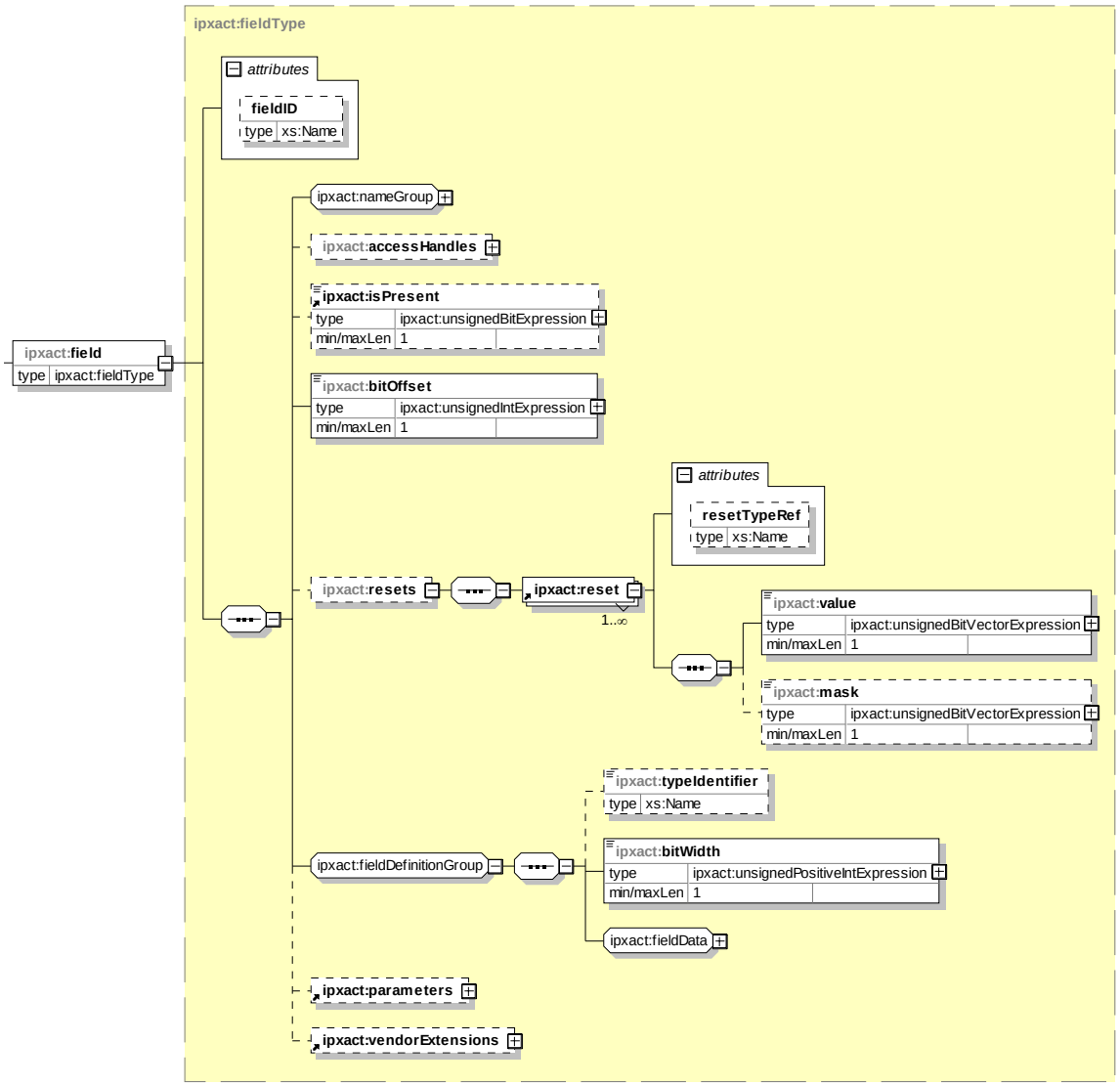
c) ***registerData*** group contains information about the grouping of bits into registers and fields. See 6.11.1.

6.11.8 Register bit fields

6.11.8.1 Schema

The following schema details the information contained in the **field** element, which may appear as an element inside the **register** element. This element describes a bit field of a register.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



6.11.8.2 Description

A **field** element of a **register** describes a smaller bit field of a register. **field** contains the following elements.

a) **fieldID** (optional; type: ***Name***) is an attribute specifying a unique ID for a field within a component.

b) ***nameGroup*** group is defined in C.12. The **name** element shall be unique within the containing **register** element.

c) **accessHandles** (optional) specifies view-dependent naming for this field within the corresponding RTL. See C.1.

d) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

e) **bitOffset** (mandatory; type: ***unsignedIntExpression*** (see C.3.7)) describes the offset (from bit `0` of the register) where this bit field starts.

f) **resets** (optional) describes the reset values of the field. Each **reset** element defines the reset value for a given reset type. **reset** has the following subelements.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

1) The **resetTypeRef** attribute (optional; type: ***Name***) identifies the type of reset being defined. If specified, this should correspond to a user-defined reset in the **resetTypes** element (see 6.19). If not specified, the default reset type is **HARD**.

2) **value** (mandatory; type: ***unsignedBitVectorExpression*** (see C.3.6)) contains the actual reset value.

3) **mask** (optional; type: ***unsignedBitVectorExpression*** (see C.3.6)) defines which bits of the register field have a known reset value.

   A `1` bit in the **mask** means the corresponding bit of the register field has a known reset value; a `0` bit means it does not. All bits of the **value** that correspond to `0` bits of the **mask** are ignored. The absence of a mask element is equivalent to a mask of the same size as the register field consisting of all `1` bits.

g) ***fieldDefinitionGroup*** group describes additional elements for a field.

1) **typeIdentifier** (optional; type: ***Name***) indicates multiple fields elements with the same **typeIdentifier** in the same description contain the exact same information for the elements in ***fieldDefinitionGroup***.

2) **bitWidth** (mandatory; type: ***unsignedPositiveIntExpression*** (see C.3.9)) is the width of the field, counting in bits.

3) ***fieldData*** group describes additional elements for a field. See 6.11.9.

h) **parameters** (optional) details any additional parameters that describe the **field** for generator usage. See C.18.

i) **vendorExtensions** (optional) adds any extra vendor-specific data related to this field. See C.24.

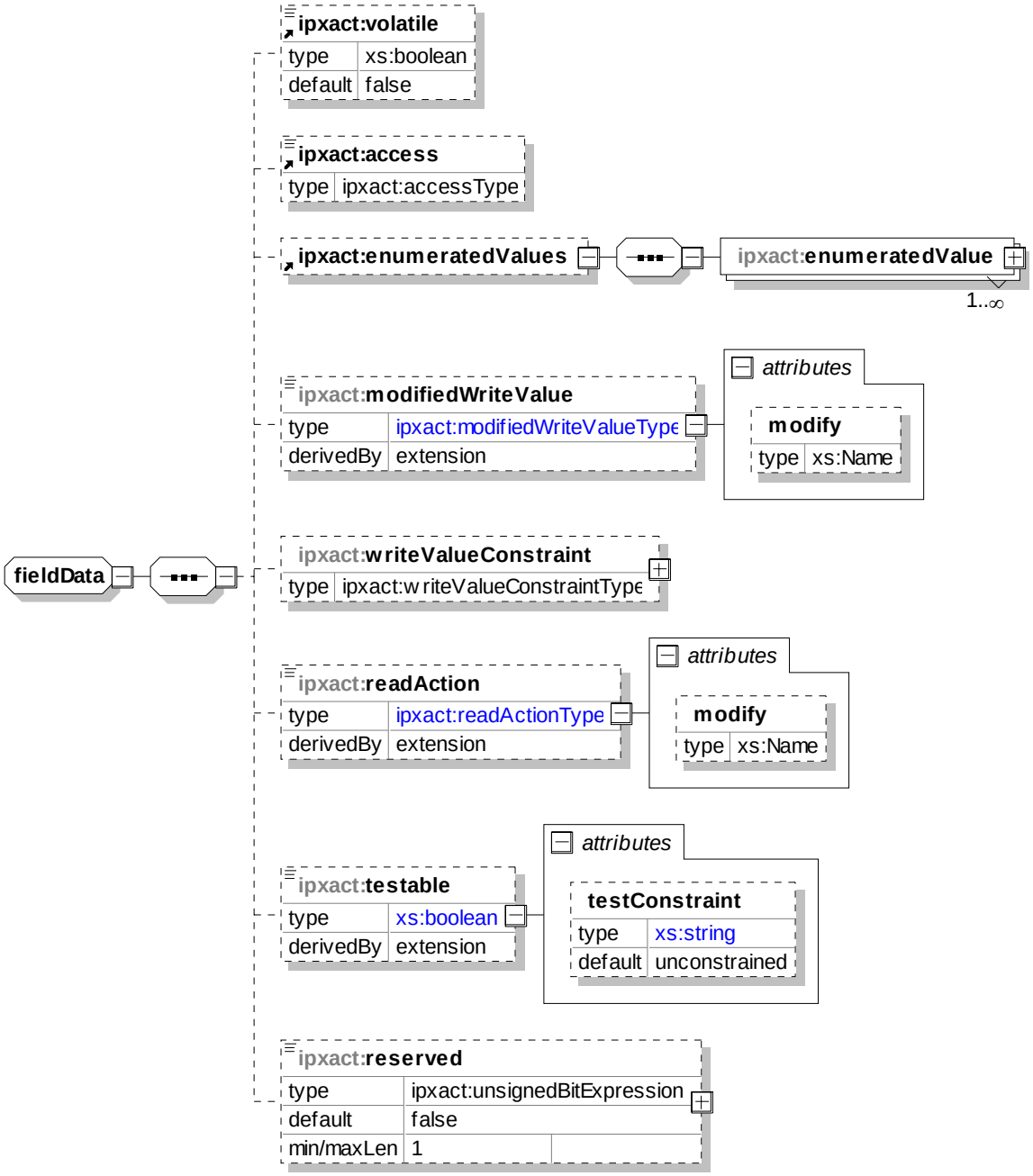
See also: SCR 7.2, SCR 7.4, SCR 7.9, SCR 7.10, SCR 7.11, SCR 7.12, SCR 7.21, and SCR 8.19.

6.11.9 Field data group

6.11.9.1 Schema

The following schema details the information contained in the ***fieldData*** group, which is contained inside the **field** element. This group describes the optional properties of the ***fieldData*** group definition.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



6.11.9.2 Description

The ***fieldData*** group contains the following elements.

a) **volatile** (optional; type: ***boolean***; default: **false**) when **true** indicates in the case of a write followed by read, or in the case of two consecutive reads, there is no guarantee as to what is returned by the read on the second transaction or that this return value is consistent with the write or read of the first transaction. The element implies there is some additional mechanism by which this field can acquire new values other than by reads/writes/resets and other access methods known to IP-XACT. If this element is not present, it is presumed to be **false**.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

b) **access** (optional) indicates the accessibility of the field. If this is not present, the **access** is inherited from the containing **register**. This element can take one of the following values.

1) **read-write**: Both read and write transactions may have an effect on this field. Write transactions may affect the contents of the field and read transactions return a value related to the values in the field.

2) **read-only**: A read transaction to this address returns a value related to the values in the field. A write transaction to this field has undefined results.

3) **write-only**: A write transaction to this address affects the contents of the field. A read transaction to this field has undefined results.

4) **read-writeOnce**: Both read and write transactions may have an effect on this field. Only the first write transaction, after power up, may affect the contents of the field and read transactions return a value related to the values in the field.

5) **writeOnce**: Only the first write transaction, after power up, to this address affects the contents of the field. A read transaction to this field has undefined results.

c) **enumeratedValues** (optional) describes a name for different values of a field. See 6.11.10.

d) **modifiedWriteValue** (optional) element to describe the manipulation of data written to a field. The value shall be one of **oneToClear**, **oneToSet**, **oneToToggle**, **zeroToClear**, **zeroToSet**, **zeroToToggle**, **clear**, **set**, or **modify**. If the **modifiedWriteValue** element is not specified, the value written to the field is the value stored in the field.

oneToClear means in a bitwise fashion each write data bit of a one shall clear (set to zero) the corresponding bit in the field.

oneToSet means in a bitwise fashion each write data bit of a one shall set (set to one) the corresponding bit in the field.

oneToToggle means in a bitwise fashion each write data bit of a one shall toggle the corresponding bit in the field.

zeroToClear means in a bitwise fashion each write data bit of a zero shall clear (set to zero) the corresponding bit in the field.

zeroToSet means in a bitwise fashion each write data bit of a zero shall set (set to one) the corresponding bit in the field.

zeroToToggle means in a bitwise fashion each write data bit of a zero shall toggle the corresponding bit in the field.

clear means after a write operation all bits in the field are cleared (set to zero).

set means that after a write operation all bits in the field are set (set to one).

modify means that after a write operation all bits in the field may be modified in an undefined way. In this situation, the **modify** attribute can be set to a user-defined value to provide additional detail.

e) **writeValueConstraint** (optional) describes a set of constraint values that are the only values that can be written to this field. If **writeValueConstraint** is not present, no constraint values are defined for this field. See 6.11.11.

f) **readAction** (optional) describes an action that happens to a field after a read operation happens. **clear** indicates the field is cleared after a read operation. **set** indicates the field is set after a read operation. **modify** indicates the field is modified in some way after a read operation. If **readAction** not specified, the field is not modified after a read operation. When set to the value `modify`, the **modify** attribute can be set to provide a user-defined enumeration providing additional information about the type of the modification.

g) **testable** (optional; type: ***boolean***) defines if the field is testable by a simple automated register test. If this is not present, **testable** is presumed to be **true**.

testConstraint (optional; type: ***string***; default: **unConstrained**) attribute defines the constraint for the field during a simple automated register test.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

unConstrained indicates there are no restrictions on the data that may be written or read from the field. **restore** indicates the field's value shall be restored to the original value before accessing another register. **writeAsRead** indicates the field shall only be written to a value just previously read from the field. **readOnly** indicates the field shall only be read.

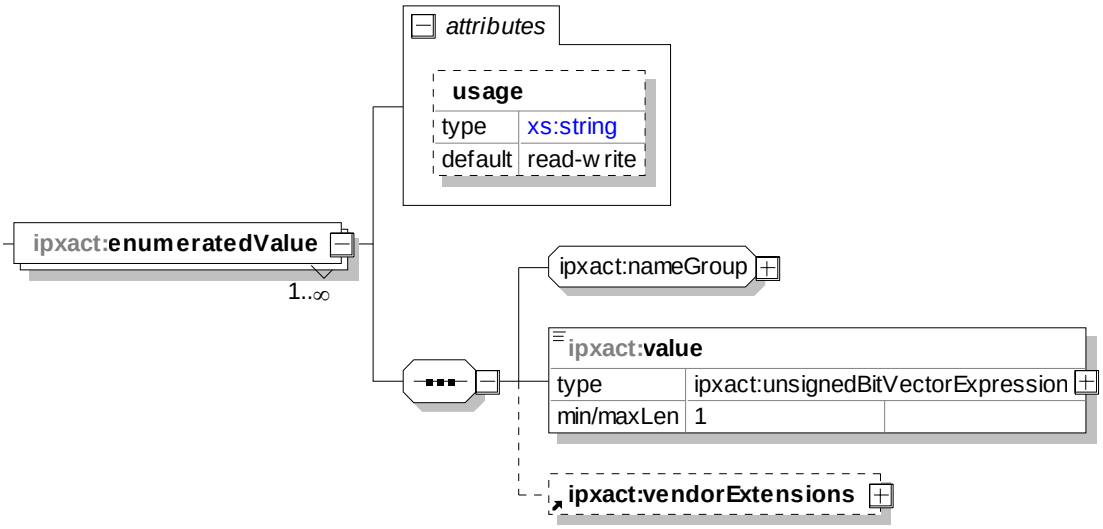
h) **reserved** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **false**) indicates this **field** is reserved. No specific definition or interpretation of the meaning of **reserved** is made.

See also: SCR 7.2, SCR 7.4, SCR 7.9, SCR 7.10, SCR 7.11, SCR 7.12, SCR 7.16, and SCR 7.17.

6.11.10 Enumeration values

6.11.10.1 Schema

The following schema details the information contained in the **enumeratedValues** element, which may appear as an element inside the **field** element.



6.11.10.2 Description

The **enumeratedValue** element describes the name and values pairs for the given field.

a) **usage** (optional; type: ***string***; default: **read-write**) attribute defines the software access condition under which this name value pair is valid. Possible values are **read**, **write**, and **read-write**.

b) ***nameGroup*** group is defined in C.12. All **name** elements shall be unique within the containing **enumeratedValues** element.

c) **value** (mandatory; type: ***unsignedBitVectorExpression*** (see C.3.6)) defines the value to assign to the specified name.

d) **vendorExtensions** (optional) adds any extra vendor-specific data related to this enumeration. See C.24.

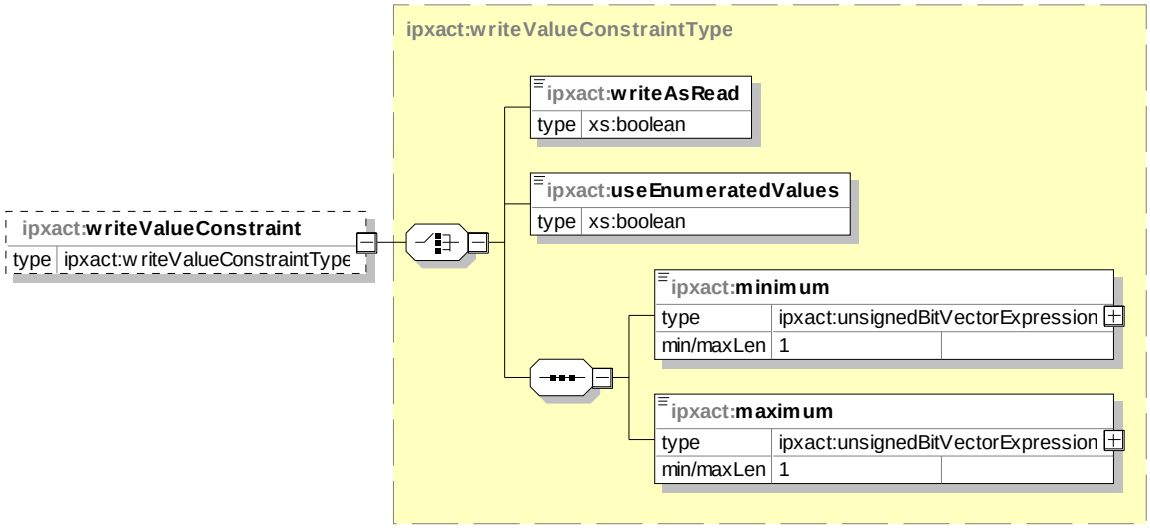
See also: SCR 7.10, SCR 7.11, SCR 7.18, and SCR 7.19.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

6.11.11 Write value constraint

6.11.11.1 Schema

The following schema details the information contained in the **writeValueConstraint** element, which may appear as an element inside the **field** element.



6.11.11.2 Description

The **writeValueConstraint** element describes a set of constraint values that are the only values that can be written to this field. If **writeValueConstraint** is not present, the legal values for this field are not defined. **writeValueConstraint** is one of the following.

a) **writeAsRead** (type: ***boolean***) if **true** implies the only legal value to write to this field is a value previously read from this field. If **writeAsRead** is not present, it is presumed to be **false**.

b) **useEnumeratedValues** (type: ***boolean***) if **true** implies the only legal values to write to this field are the values specified in the **useEnumeratedValues** element for this field. If **useEnumeratedValues** is not present, it is presumed to be **false**.

c) **writeValueConstraint** can also be both of the **minimum** and **maximum** elements.

   1) **minimum** (type: ***unsignedBitVectorExpression*** (see C.3.6)) contains the minimum value that may be written to this field.

   2) **maximum** (type: ***unsignedBitVectorExpression*** (see C.3.6)) contains the maximum value that may be written to this field.

See also: SCR 7.10 and SCR 7.11.

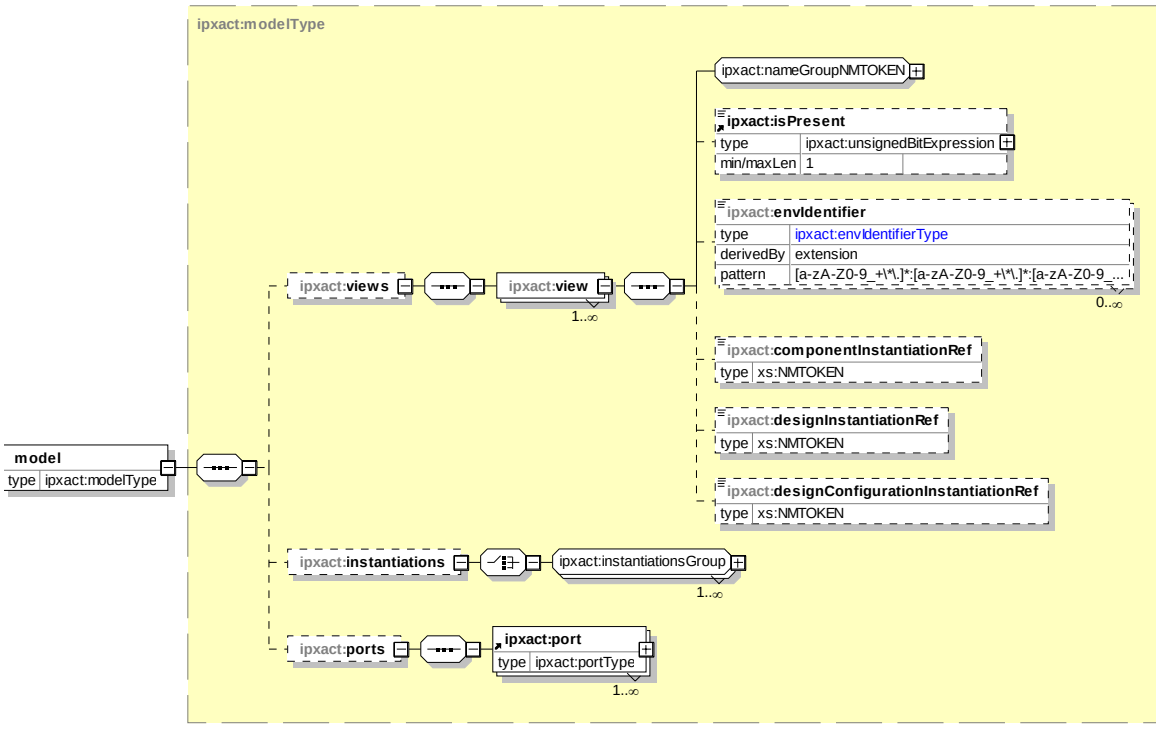
Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

6.12 Models

6.12.1 Model

6.12.1.1 Schema

The following schema details the information contained in the **model** element, which may appear as an element inside the **component** element.



6.12.1.2 Description

The **model** element describes the views, instantiations, and ports of a component. A **model** element may contain the following.

a) **views** (optional) contains a list of all the views for this object. A component may have many different views. Each view can reference a component instantiation, a design instantiation, and a design configuration instantiation. A *component instantiation* may describe the source hardware module/entity with its pin interface. A *design instantiation* references an IP-XACT design representation. A *design configuration instantiation* configures an IP-XACT design configuration representation.

   A **views** element describes an unbounded set of **view** elements. Each **view** element specifies a representation level of a component. It contains the following elements.

   1) ***nameGroupNMToken*** group is detailed in C.13. The **name** elements shall be unique within the containing **views** element.

   2) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

   3) **envIdentifier** (optional) designates and qualifies information about how this model view is deployed in a particular tool environment. The format of the element is a string with three fields

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

separated by colons [:] in the format of *Language*:*Tool*:*VendorSpecific*. The regular expression that is used to check the string is [A-Za-z0-9_+*\.]*:[A-Za-z0-9_+*\.]*:[A-Za-z0-9_+*\.]* The sections are:

i) *Language* indicates this view may be compatible with a particular tool, but only if that language is supported in that tool, e.g., different versions of some simulators may support two or more languages. In some cases, knowing the tool compatibility is not enough and may be further qualified by language compatibility, e.g., a compiled HDL model may work in a VHDL-enabled version of a simulator, but not in a SystemC-enabled version of the same simulator.

ii) *Tool* indicates this view contains information that is suitable for the named tool. This might be used if this view references data that is tool-specific and would not work generically, e.g., HDL models that use simulator-specific extensions.

Vendors shall publish lists of approved tool identification strings. These strings shall contain the tool name, as well as the company's domain name, separated by dots. Some examples of well-formed tool entries are:

```
designcompiler.synopsys.com
ncsim.cadence.com
modelsim.mentor.com
```

This field can alternatively indicate generic tool family compatibility, including `*Simulation` and `*Synthesis`. To support transportability of created data files, it is important to use the published, generally recognized, tool designation when referencing a tool. See IP-XACT standard tool names for **envIdentifier** [B15].

iii) *VendorSpecific* can be used to further qualify tool and language compatibility. This can be used to indicate additional processing information may be required to use this model in a particular environment. For instance, if the model is a SWIFT simulation model, the appropriate simulator interface may need to be enabled and activated.

Any or all of the **envIdentifier** fields may be used. Where there are multiple environments for which a particular **view** is applicable, multiple **envIdentifier** elements can be listed.

4) **componentInstantiationRef** (optional; type: ***NMTOKEN***) specifies a name that references a component inside the **instantiations** element.

5) **designInstantiationRef** (optional; type: ***NMTOKEN***) specifies a name that references a design inside the **instantiations** element.

6) **designConfigurationInstantiationRef** (optional; type: ***NMTOKEN***) specifies a name that references a design configuration inside the **instantiations** element.

b) **instantiations** (optional) specifies the component, design, and design configuration instantiations for all views (as an ***instantiationsGroup***). See 6.12.2.

c) **ports** (optional) contains the list of ports for this object. A ports is an external connection from the object. See 6.12.7.

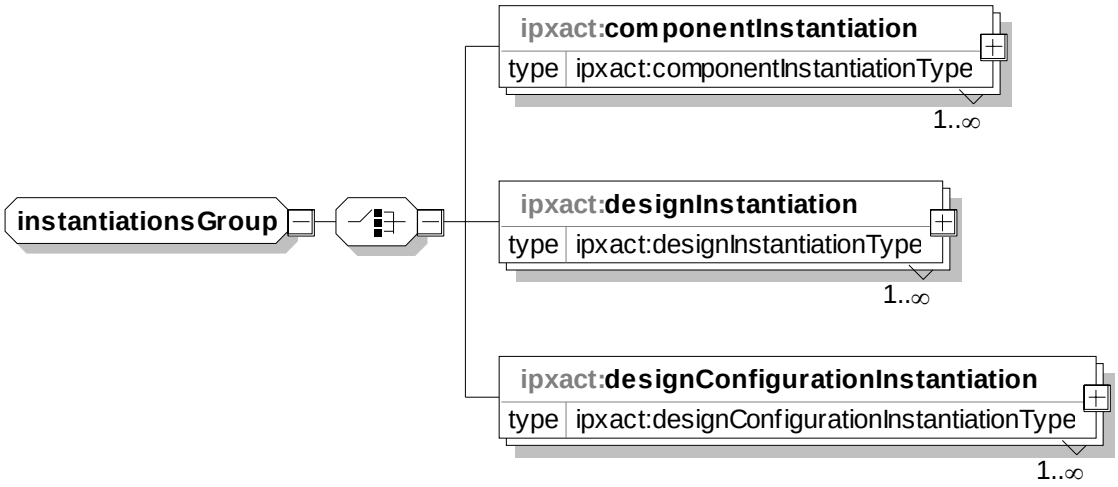
See also: SCR 1.16, SCR 1.17, SCR 5.8, and SCR 6.27.

6.12.2 instantiationsGroup

6.12.2.1 Schema

The following schema details the information contained in the ***instantiationsGroup*** group, which is contained inside an **instantiations** element inside a **model** element.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



6.12.2.2 Description

An ***instantiationsGroup*** group specifies the component, design, and design configuration instantiation for a particular view. There can be multiple occurrences of ***instantiationsGroup*** (see 6.12.1.1) and each ***instantiationsGroup*** can have multiple elements from any branch. Each occurrence shall contain one or more of the following elements.

a) **componentInstantiation** (type: ***componentInstantiationType***) specifies information concerning the instantiation of this component. See 6.12.3.

b) **designInstantiation** (type: ***designInstantiationType***) specifies information concerning the internal (design) connectivity of this component, including a reference to an IP-XACT **design** document. See 6.12.4.

c) **designConfigurationInstantiation** (type: ***designConfigurationInstantiationType***) specifies information concerning the internal (design) configuration of this component, including a reference to an IP-XACT **designConfiguration** document defining the configured hierarchy of this component. See 6.12.5.

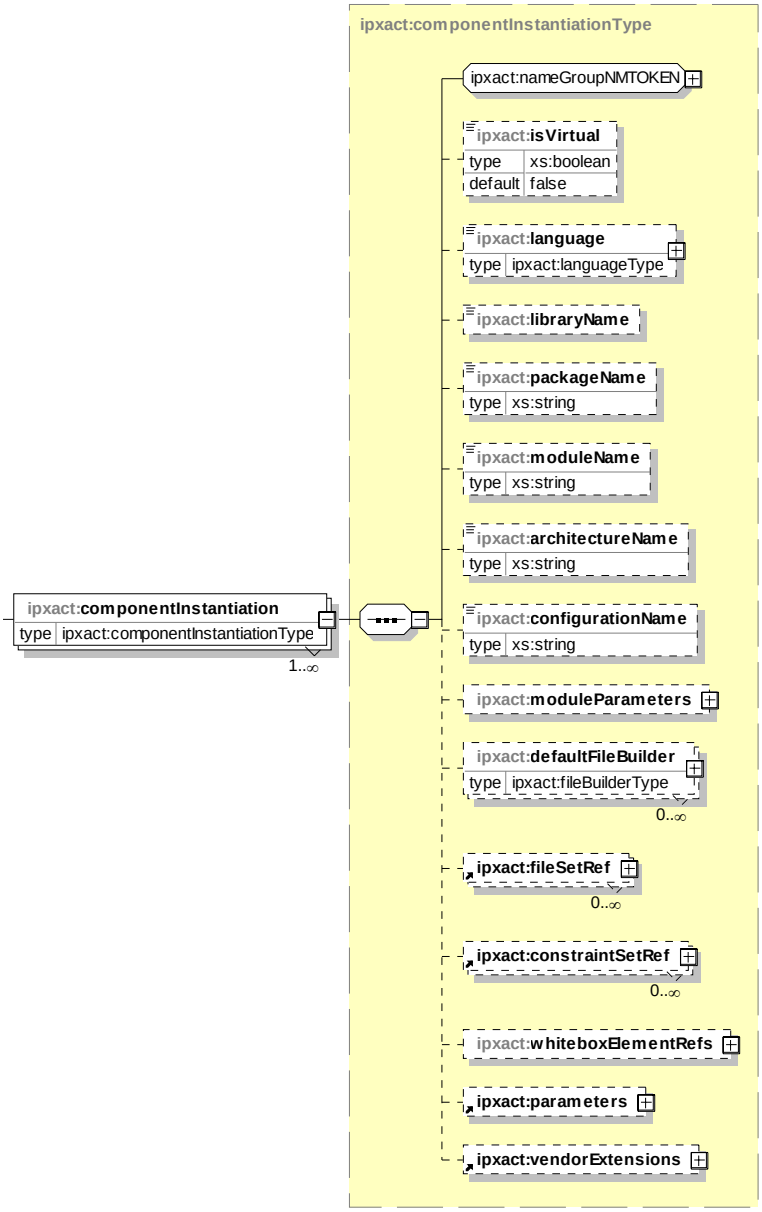
See also: SCR 5.14, SCR 5.15, and SCR 5.23.

6.12.3 componentInstantiation

6.12.3.1 Schema

The following schema details the information contained in the ***componentInstantiation*** element, which is the type of the component element within a **model**/**instantiations** element.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



6.12.3.2 Description

An ***componentInstantiation*** element details how a specific component instance is associated with a view. It has the following elements.

a) ***nameGroupNMToken*** group is detailed in C.13. The **name** elements shall be unique within the containing **instantiations** element.

b) **isVirtual** (optional; type: ***boolean***; default: **false**) when **true**, indicates this component should not be netlisted.

c) **language** (optional) specifies the HDL used for a specific instantiation, for example, `verilog` or `vhdl`. The **language** element is of type ***languageType***, which is a token string along with an optional boolean attribute **strict**. If **true**, the language shall be strictly enforced. If this attribute is not present, its effective value is **false**.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

d) **libraryName** (optional) is a string which specifies the library name in which the model should be compiled. If the **libraryName** element is not present, then its value defaults to `work`.

e) **packageName** (optional; type: ***string***) is a string which describes the VHDL package containing the interface of the model. If the **packageName** element is not present, then its value defaults to the component VLNV name concatenated with the postfix `_cmp_pkg`, which stands for component package.

f) **moduleName** (optional; type: ***string***) is a string which describes the Verilog, SystemVerilog, or SystemC module name or the VHDL entity name. If the **moduleName** element is not present, then its value defaults to the component VLNV name.

g) **architectureName** (optional; type: ***string***) is a string which describes the VHDL architecture name. If the **architectureName** element is not present, then its value defaults to `rtl`.

h) **configurationName** (optional; type: ***string***) is a string which describes the Verilog, SystemVerilog, or VHDL configuration name. If the **configurationName** element is not present, then its value defaults to the design configuration VLNV name of the design configuration referenced in the view or, if there is no such reference in the view, to the component VLNV name concatenated with the postfix `_rtl_cfg`.

i) **moduleParameters** (optional) specifies a parameter name-value pair container. See 6.12.6.

j) **defaultFileBuilder** (optional) is an unbounded list of default file builder options for the **fileSets** referenced in this **view**. This contains all the same elements as the element **fileBuilder**. See 6.15.4.

k) **fileSetRef** (optional) is an unbounded list of references to **fileSet name**s within the containing document or another document referenced by the **VLNV**. See C.7.

l) **constraintSetRef** (optional) is an unbounded list of references to constraint sets, valid timing constraints for a view. **constraintsSets** are only defined for **wire** style **ports**. The **constraintSetRef** element is of type ***NMTOKEN***. See 6.12.12.

m) **whiteboxElementRefs** (optional) contains references to white box elements of a component that are valid for this view. If the view contains an implementation of any of the white box elements for the component, the **view** section shall include a reference to that **whitebox** element, with a string providing a language-dependent path to enable the DE to access the **whitebox** element. See 6.17.

n) **parameters** (optional) details any additional parameters that describe the instantiation. See C.18.

o) **vendorExtensions** (optional) adds any extra vendor-specific data related to the instantiation. See C.24.

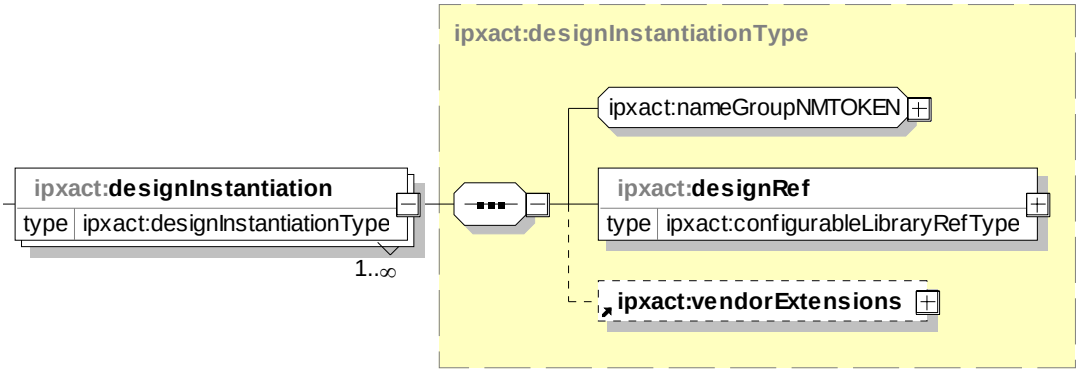
See also: SCR 6.28.

6.12.4 designInstantiation

6.12.4.1 Schema

The following schema details the information contained in the ***designInstantiation*** element, which is the type of the design element within a **model**/**instantiations** element.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



A ***designInstantiation*** element details how a specific design instance is associated with a view. It has the following elements.

a) ***nameGroupNMToken*** group is detailed in C.13. The **name** elements shall be unique within the containing **instantiations** element.

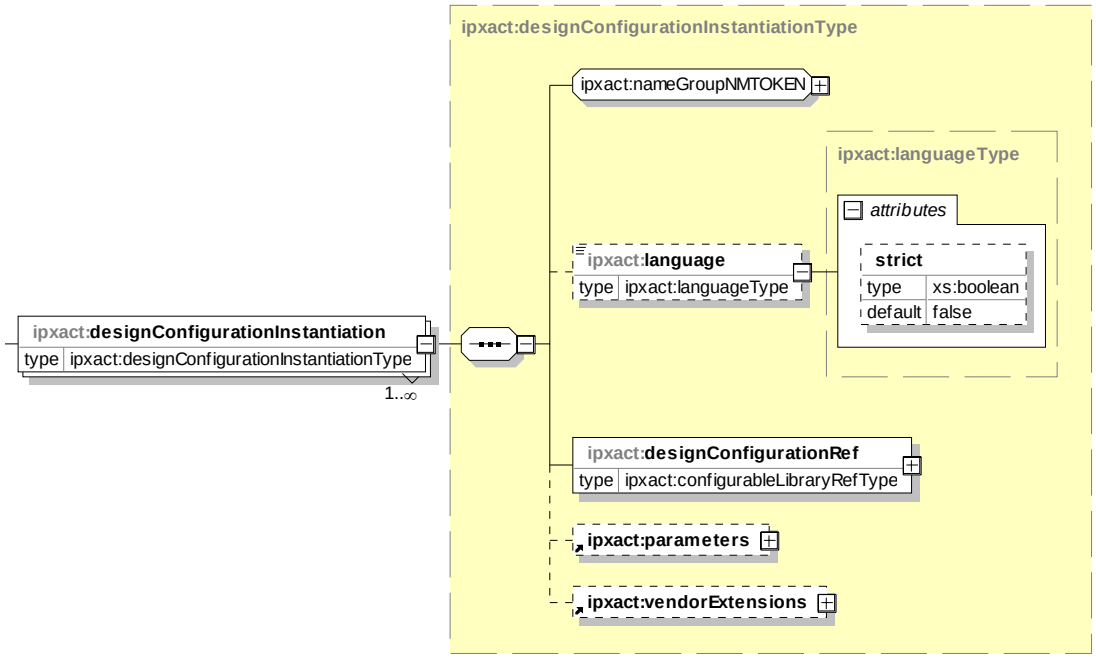
b) **designRef** (mandatory; type: ***configurableLibraryRefType*** (see C.6)) specifies the design description and configuration values to applied to the design description.

c) **vendorExtensions** (optional) adds any extra vendor-specific data related to the view. See C.24.

6.12.5 designConfigurationInstantiation

6.12.5.1 Schema

The following schema details the information contained in the ***designConfigurationInstantiation*** element definition, which is the type of the design configuration element within a **model/instantiations** element.



Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

A ***designConfigurationInstantiation*** element details how a specific design configuration instance is associated with a view. It has the following elements.

a) ***nameGroupNMToken*** group is detailed in C.13. The **name** elements shall be unique within the containing **views** element.

b) **language** (optional) specifies the HDL used for a specific instantiation, for example, `verilog` or `vhdl`. The **language** element is of type ***languageType***, which is a token string along with an optional boolean attribute **strict** (type: ***boolean***; default: **false**). If **true**, the language shall be strictly enforced. If this attribute is not present, its effective value is **false**.

c) **designConfigurationRef** (mandatory; type: ***configurableLibraryRefType*** (see C.6)) specifies the design configuration description.

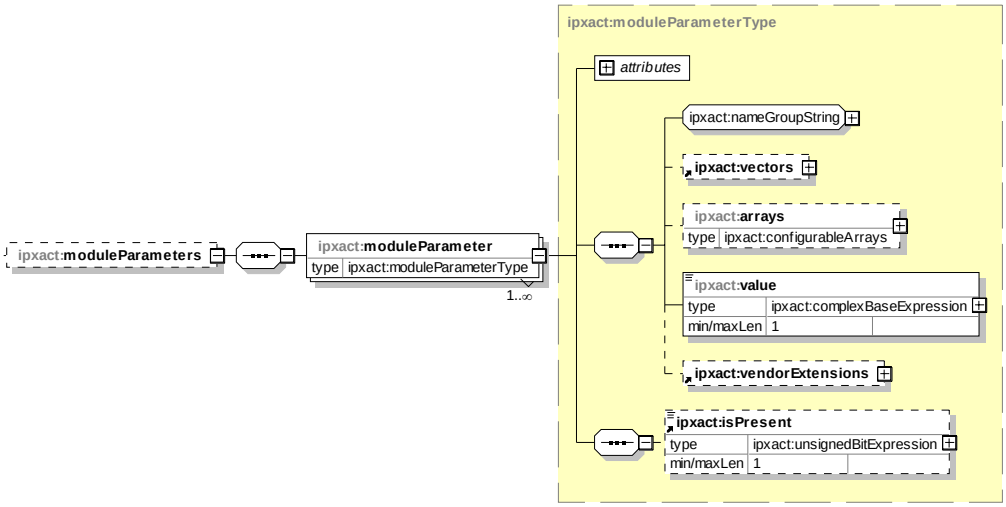
d) **parameters** (optional) details any additional parameters that describe the instantiation. See C.18.

e) **vendorExtensions** (optional) adds any extra vendor-specific data related to the instantiation. See C.24.

6.12.6 Module parameters

6.12.6.1 Schema

The following schema details the information contained in the **moduleParameters** element, which may appear as an element inside a **component**/**model**/**instantiations**/**component** element.



6.12.6.2 Description

A **moduleParameters** element contains an unbounded list of **moduleParameter** elements. The **moduleParameter** element describes a name-value pair used to configure the model of this component instance. A **moduleParameter** contains the following elements and attributes.

a) Attributes used to constrain the parameter value and enable presentation to the user are defined in C.19. **moduleParameter** also has the following attributes.

1) **dataType** (optional; type: ***string***) attribute specifies the data type as it pertains to the language of the model. This definition is used to define the type for component declaration and as such has no semantic meaning. For example, SystemC could be `int`, `double`, `char*`, etc. For VHDL, this could be `std_logic`, `std_logic_vector`, `integer`, etc.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

2) **usageType** (optional; default: **nontyped**) attribute specifies how this parameter is used in different modeling languages: **nontyped** and **typed**. See 6.12.6.2.1.

b) ***nameGroupString*** group is defined in C.16.

c) **vectors** (optional) specifies dimensions required to define a vectored value type (see C.23).

d) **arrays** (optional; type: ***configurableArrays*** (see C.4)) specifies dimensions required to define an arrayed value type.

e) **value** (mandatory; type: ***complexBaseExpression*** (see C.3)) contains the value of this module parameter. This needs to meet the type requirements defined within the attributes on this element (see C.19).

f) **vendorExtensions** (optional) adds any extra vendor-specific data related to the port. See C.24.

g) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

See also: SCR 5.2.

6.12.6.2.1 Typed and non-typed parameters classification

There are two categories of parameters: typed and non-typed.

The *typed* parameters (or declaration parameters) appear in object-oriented (OO) languages such a C++/ SystemC or SystemVerilog.

In C++/SystemC, these are named `Class` template parameters. Templates can be used to develop a generic class prototype (specification), which can be instantiated with different data types. This is very useful when the same kind of class is used with different data types for individual members of the class. Parameterized types are used as data types and then a class can be instantiated, i.e., constructed and used by providing arguments for the parameters of the class template. A class template is a specification of how a class should be built (i.e., instantiated) given the data type or values of its parameters.

`Class` template parameters can have default arguments, which are used during class template instantiation when arguments are not provided. Because the provided arguments are used starting from the far left parameter, default arguments should be provided for the right-most parameters.

Example 1

```
template <typename T>
class FIFO {
          FIFO();
          T pull();
          void push(T &x);
};
```

In SystemVerilog, typed parameters are named type parameters. Type parameters can be used in SystemVerilog classes, interfaces, or modules to provide the basic function of C++ templates.

Example 2

```
typedef bit[32] DataT;
interface FIFO #(type T);
          Method T pull();
          Method push (T x);
endinterface: FIFO
```

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

The generic *non-typed* parameters (or initialization parameters) appear in all languages (procedural or OO) and in particular in VHDL, Verilog, SystemC, and SystemVerilog. A non-typed parameter is like an ordinary (function-parameter) declaration. In SystemC, it represents a constant in a class template definition or a parameter in a class constructor, i.e., this can be determined at compilation time. In VHDL, it is represented by generics. In Verilog or SystemVerilog, it is represented by parameters.

Example 3

Here is an example of non-typed parameters usage on a simple `GCD` model expressed in various languages.

VHDL

```
entity GCD is
generic (Width: natural);
port (
Clock,Reset,Load: in std_logic;

                    A,B:      in unsigned(Width-1 downto 0);
                    Done:     out std_logic;
                    Y:        out unsigned(Width-1 downto 0));
end entity GCD;
```

(System)Verilog

```
module GCD (Clock, Reset, Load, A, B, Done, Y);
parameter Width = 8;
                    input     Clock, Reset, Load;
                    input     [Width-1:0] A, B;
                    output    Done;
                    output    [Width-1:0] Y;
…
endmodule
```

SystemC

```
template <unsigned int Width = 8>
SC_MODULE (GCD) {
                    sc_in<bool> Clock, Reset, Load;
                    sc_in<sc_uint<Width> >a, b;
                    sc_out<bool> Done;
                    sc_out<sc_uint<Width> > y;
                    …
}
```

These two kinds of parameters (typed and non-typed) can be combined to model complex IP modules.

Example 4

In SystemC:

```
template <typename T> // type parameter
class testModule : public sc_module {
public:
                    testModule(sc_module_name modnamemodname, string
   portname) :
                        // non type parameters
```


Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
                        sc_module(modname),
                        testport(portname) {…}
                        sc_port<T> testport;
};
```

In a top SystemC netlist design, such a class is instantiated as follows.

```
testModule<bool> test("myModuleName","port1");
```

6.12.6.2.2 Generic parameters mapping in different languages

Table 4 summarizes the two kinds of parameters (initialization and declaration) expressed in the four most commonly used hardware languages.

Table 4—Parameter mappings

Language	Non-typed parameters (initialization)	Typed parameters (declaration)
VHDL	`generics`	NA
Verilog	`parameter`	NA
SystemC	`constructor`	`template` (`constant` or `variable` type)
SystemVerilog	`parameter`	`parameter`

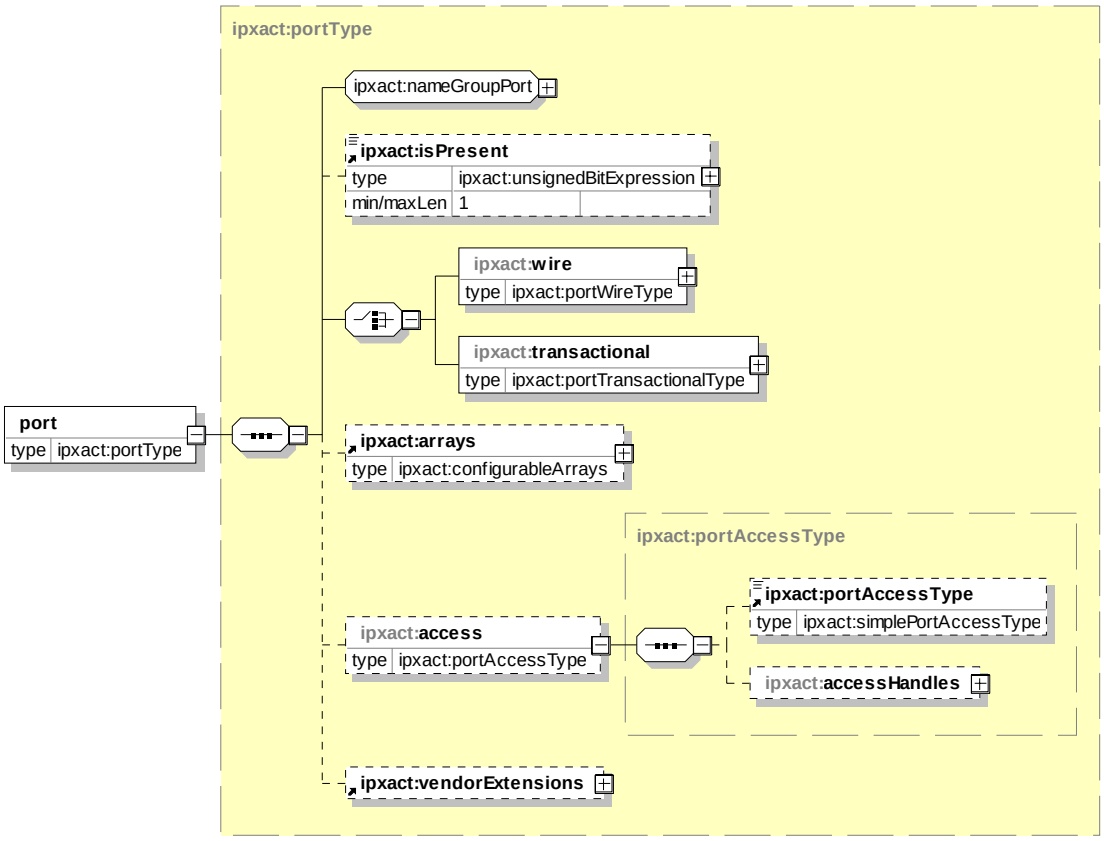
A *declaration parameter* (e.g., `int`) shall be used when declaring an IP instance in a top netlist (e.g., `myIP int myIntIP;`). An *initialization parameter* (e.g., `myName`) shall be used when initializing the instance of that IP (e.g., `myIntIP("myName");`).

6.12.7 Component ports

6.12.7.1 Schema

The following schema defines the information contained in the **ports** element, which may appear within a **component**.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



6.12.7.2 Description

The **ports** element defines an unbounded list of **port** elements. Each **port** element describes a single external port on the component.

a) ***nameGroupPort*** group is defined in C.15. The **name** elements shall be unique within the containing **ports** element.

b) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

c) Each **port** shall be described as a **wire** or **transactional** port.

1) **wire** defines ports that transport purely binary values. Values may be aggregated using vectors and/or arrays. See 6.12.8.

2) **transactional** defines all other styles of ports, typically used for TLM. See 6.12.17.

d) **arrays** (optional type: ***configurableArrays*** (see C.4)) specifies dimensions for a port defined as an array.

e) **access** (optional) defines the access for a port.

1) **portAccessType** (optional; default: **ref**) indicates to a netlister how to access the port. The **portAccessType** has one of two possible values **ref** or **ptr**. If **ref**, a netlister should access the port directly (i.e., by reference) and, if **ptr**, it should access the port with a pointer.

2) **accessHandles** (optional) indicates to a netlister the method to be used to access the object representing the port when references need to be done using something other than the simple port name. This is typically a function call. The **accessHandles** element contains a list of **accessHandle** elements of type ***leafAccessHandle***. See C.1.4.

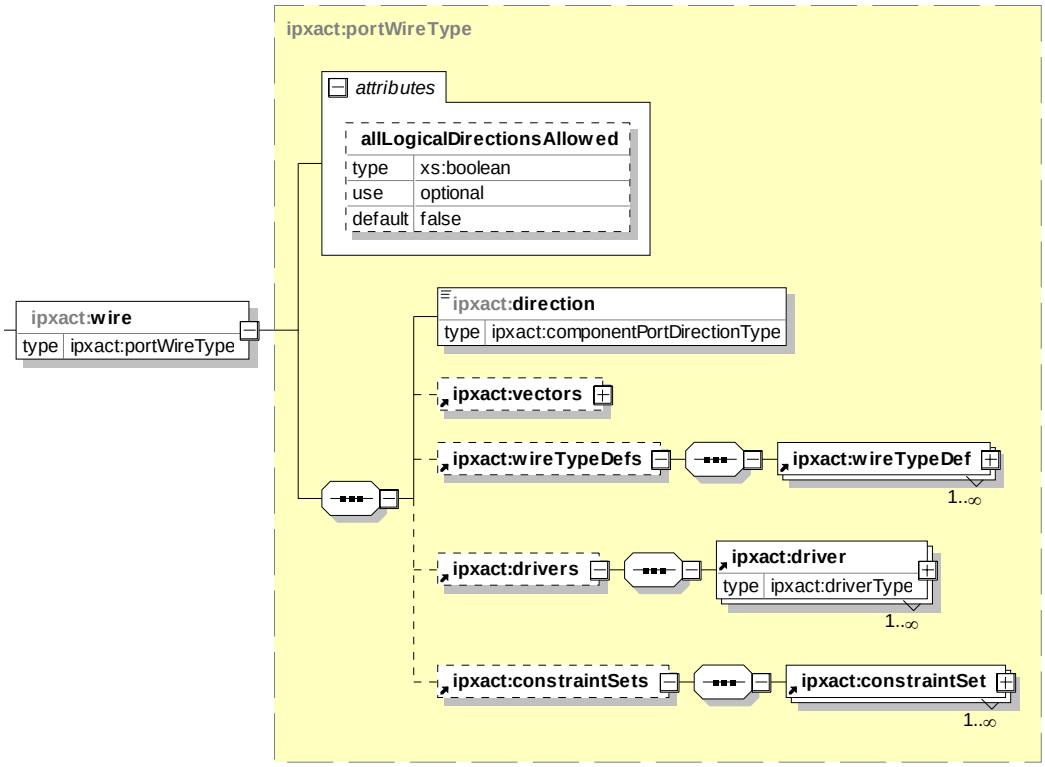
Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

f) **vendorExtensions** (optional) adds any extra vendor-specific data related to the port. See C.24.

6.12.8 Component wire ports

6.12.8.1 Schema

The following schema details the information contained in the **wire** element, which may appear as an element inside the top-level **component**/**model**/**port** element.



6.12.8.2 Description

The **wire** element describes the properties for ports that are of a wire style. A port can come in two different styles, wire or transactional. A wire port applies for all scalar types (e.g., VHDL `std_logic` and Verilog `wire`) and vectors of scalars. Scalar types in VHDL also include integer and enumeration values; however, scalars in IP-XACT only include binary values that relate to a single wire in a hardware implementation.

A wire port transports purely binary values or vectors of binary values; IP-XACT does not support tri-state or multiple strength values.

The **wire** element contains the following attributes and elements.

a) **allLogicalDirectionsAllowed** (optional; type: ***boolean***; default: **false**) attribute defines whether the port may be mapped to a port in an **abstractionDefinition** with a different direction.See 5.3.

b) **direction** (mandatory) specifies the direction of this port: **in** for input ports, **out** for output ports, and **inout** for bidirectional and tri-state ports. **phantom** can also be used to define a port that only exists on the IP-XACT component, but not on the implementation referenced from the view.

c) **vectors** (optional) specifies the dimensions for a non-scalar port, see C.23.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

d) **wireTypeDefs** (optional) describes the ports type as defined by the implementation, see 6.12.9.

e) **drivers** (optional) defines an unbounded list of driver elements, defining drivers that may be attached to this port if no other object is connected to this port. This allows the IP to define the default state of unconnected inputs. A wire style port may only define a **driver** element for a port if the direction of the port is **in** or **inout**. See also 6.12.10.

If multiple drivers are specified, the **range** element shall be used to indicate which bits are driven by which driver and no overlap bit ranges may be specified.

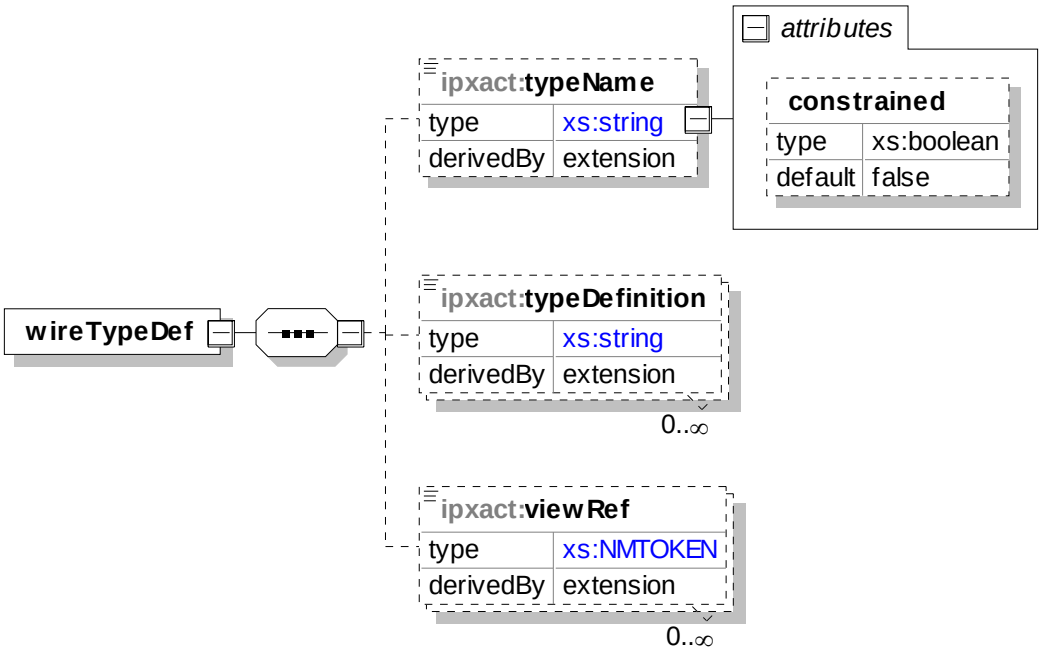
f) **constraintSets** (optional) defines multiple set of constraints on a port used for synthesis or other operations. See 6.12.14.

See also: SCR 6.5, SCR 6.6, SCR 6.7, and SCR 6.12.

6.12.9 Component wireTypeDef

6.12.9.1 Schema

The following schema details the information contained in the **wireTypeDef** element, which may appear as an element inside the **wire** element of a top-level wire style **port** element. These elements define the definition type name, where the type is defined, and which views of a component or an abstractor use this type.



6.12.9.2 Description

The **wireTypeDef** element describes the type properties for a port per view of a component or abstractor. There can be an unbounded series of **wireTypeDef**s defined for each port, allowing the type properties to be defined differently for each view. If a port does not have **wireTypeDef**s, the default **wireTypeDef** applies for each view (see Table 6).

wireTypeDef contains the following elements.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

a) **typeName** (optional; type: ***string***) defines the name of the type for the port. For VHDL, some typical values would be `std_logic` and `std_ulogic`.

constrained (optional; type: ***boolean***; default: **false**) attribute indicates whether or not the number of bits in the type declaration is fixed or not. If the number of bits is fixed (**constrained** is **true**), the bit indices are not required when referencing the type. When **constrained** is **false**, bit indices are required on all references to the type definition. See 6.12.9.2.1 and 6.12.9.2.1.

b) **typeDefinition** (optional; type: ***string***) contains a language-specific reference to where the given type is actually defined. Table 5 shows some examples. There can be multiple **typeDefinition**s for each port.

Table 5—typeDefinition examples

Language	Meaning
VHDL	"Use" statement text (`IEEE.std_logic_1164.all`).
Verilog	Nothing needed, no meaning.
SystemC	Include file name (`systemc.h`).
SystemVerilog	Include file name (if the name does not contain a `:`); import package name (if the name contains a `:`).

c) **viewRef** (optional; type: ***NMTOKEN***) specifies the views to which the **wireTypeDef** applies. See C.26.

6.12.9.2.1 Constrained and unconstrained array types

A *constrained array type* is a type for which the indices of the array have been specified in the definition. An *unconstrained array type* is a type for which the indices of the array have not been specified in the definition. An example of the RTL and corresponding IP-XACT is shown below.

```
type BYTE is array (7 downto 0) of std_logic; -- constrained
type my_logic_vector is array (NATURAL RANGE <>) of std_logic; -- unconstrained
entity example is
  port(
    A: out BYTE;
    B: out my_logic_vector(7 downto 0)
);
end example;

<ipxact:port>
  <ipxact:name>A</ipxact:name>
  <ipxact:wire>
    <ipxact:vectors>
      <ipxact:vector>
        <ipxact:left>7</ipxact:left>
        <ipxact:right>0</ipxact:right>
      </ipxact:vector>
    </ipxact:vectors>
    <ipxact:wireTypeDefs>
      <ipxact:wireTypeDef>
        <ipxact:typeName ipxact:constrained="true">BYTE</ipxact:typeName>
        <ipxact:typeDefinition>MYLIB.MYPKG.all</ipxact:typeDefinition>
        <ipxact:viewNameRef>VHDLsimView</ipxact:viewNameRef>
```

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
        </ipxact:wireTypeDef>
      </ipxact:wireTypeDefs>
    </ipxact:wire>
  </ipxact:port>
  <ipxact:port>
    <ipxact:name>B my_logic_vector</ipxact:name>
    <ipxact:wire>
      <ipxact:vectors>
        <ipxact:vector>
          <ipxact:left>7</ipxact:left>
          <ipxact:right>0</ipxact:right>
        </ipxact:vector>
      </ipxact:vectors>
      <ipxact:wireTypeDefs>
        <ipxact:wireTypeDef>
          <ipxact:typeName ipxact:constrained="false">BYTE
          </ipxact:typeName>
          <ipxact:typeDefinition>MYLIB.MYPKG.all</ipxact:typeDefinition>
          <ipxact:viewNameRef>VHDLsimView</ipxact:viewNameRef>
        </ipxact:wireTypeDef>
      </ipxact:wireTypeDefs>
    </ipxact:wire>
  </ipxact:port>
```

6.12.9.2.2 Defaults

wireTypeDefs do not need to be defined for every view of a port. IP-XACT provides for these defaults based on the language of the view, as shown in Table 6. For those languages not shown here, no defaults can be presumed.

Table 6—View defaults

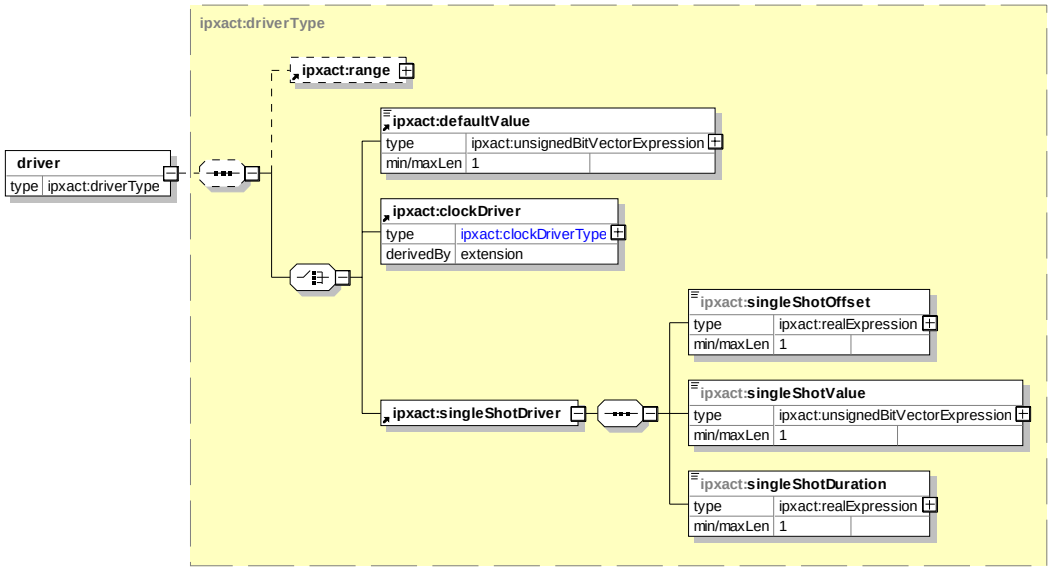
Language	Single bit	Vectors
VHDL	`std_logic`	`std_logic_vector`
Verilog	`wire`	`wire`
SystemC	`sc_logic`	`sc_lv`
SystemVerilog	`logic`	`logic`

6.12.10 Component driver

6.12.10.1 Schema

The following schema details the information contained in the **driver** element, which may appear as an element inside the **wire**/**driver** element of a top-level wire style **port** element. This element defines the type and value(s) to drive on this port when it is not connected in a design; it is only allowed on ports with the direction **in** or **inout**.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



6.12.10.2 Description

The **driver** element shall contain one of three different types of drivers (**defaultValue**, **clockDriver**, or **singleShotDriver**) that can be applied to a wire port of a component or abstractor.

a) **range** (optional) specifies the bit range of the port to which the given **driver** element is to be applied. If not specified, the **driver** applies to the entire port (see C.22).

b) A **driver** shall also contain one of the following.

1) **defaultValue** (type: ***unsignedBitVectorExpression***, see C.3.6)) specifies a static logic value for this port. The **defaultValue** shall be defined to have a bit width that is compatible with the range of the port being driven. The value shall be applied to this port when it is left unconnected, independent of being part of a **busInterface**. This value shall be over-ridden by the default value from the abstraction definition if the interface is connected.

2) **clockDriver** specifies a repeating waveform for the specified bit range of this port. See 6.12.11.

3) **singleShotDriver** specifies a non-repeating waveform for this port. When this element is contained within a non-scalar wire port, the single-shot waveform shall apply to all bits of this port. The **singleShotDriver** element contains three elements that describe the properties of the waveform. These are also depicted in Figure 10.

i) **singleShotOffset** (mandatory; type: ***realExpression*** (see C.3.1)) specifies the time delay from the start of the waveform to the transition.

This element also contains a **units** (optional) attribute for specifying the units of the time values: **ns** (the default) and **ps**, where **ns** stands for nanosecond and **ps** stands for picosecond.

ii) **singleShotValue** (mandatory; type: ***unsignedBitVectorExpression***, see C.3.6)) specifies the logic value to which the port transitions. This value is also the opposite of the value from which the waveform starts. The value of this element shall be `0` or `1`.

iii) **singleShotDuration** (mandatory; type: ***realExpression*** (see C.3.1)) specifies how long the waveform remains at the value specified by **singleShotValue**.

This element also contains a **units** (optional) attribute for specifying the units of the time values: **ns** (the default) and **ps**, where **ns** stands for nanosecond and **ps** stands for picosecond.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

See also: SCR 6.24 and SCR 12.9.

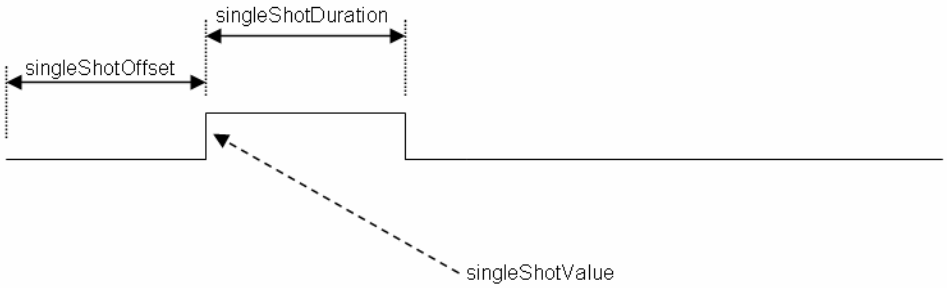


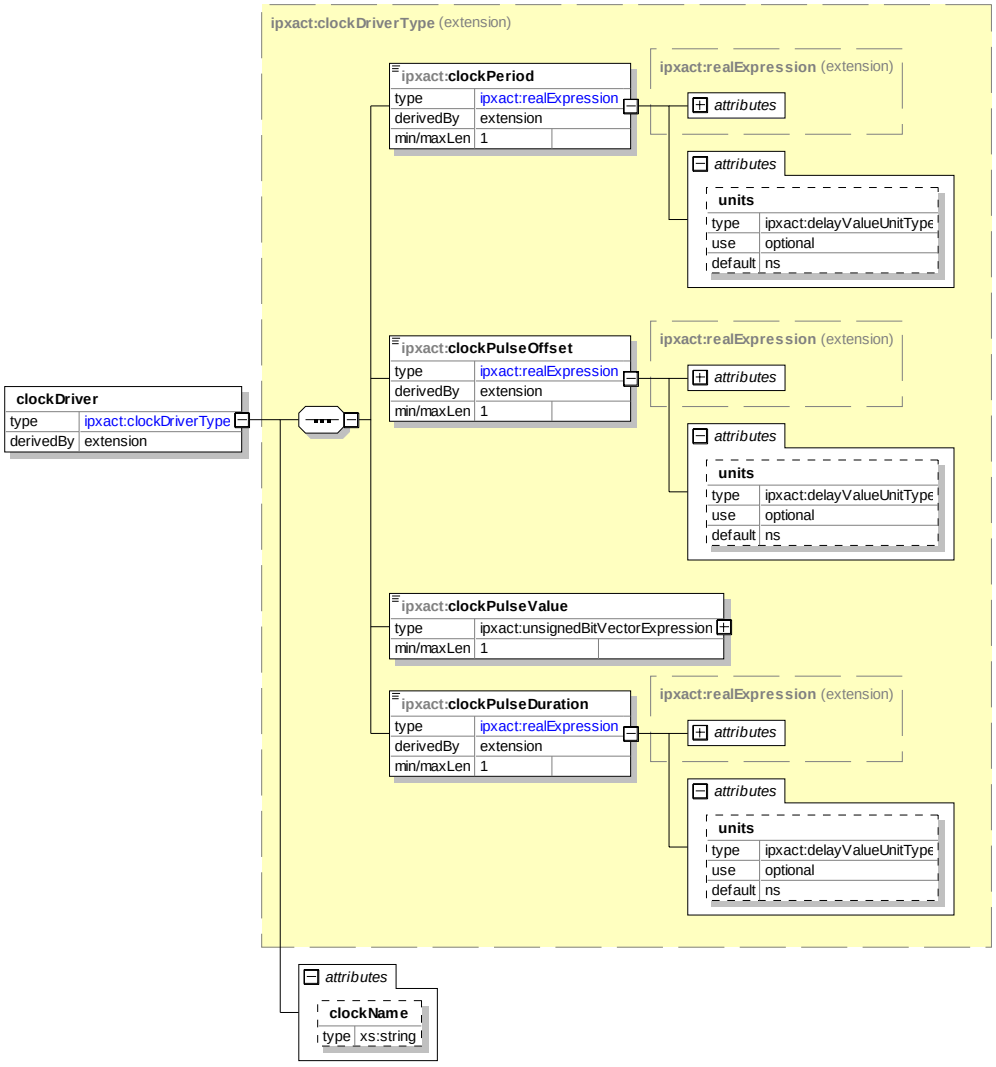
Figure 10—singleShotDriver elements

6.12.11 Component driver/clockDriver

6.12.11.1 Schema

The following schema details the information contained in the **clockDriver** element, which may appear as an element inside the top-level wire style **port**/**wire**/**driver** element. This element defines the properties of a clock waveform. When this element is contained within a non-scalar wire port, the clock waveform shall apply to all bits of this port.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



6.12.11.2 Description

The **clockDriver** element contains four elements that describe the properties of a clock waveform (see also: Figure 11). The optional **clockName** attribute (type: ***string***) is used to specify a name for the clock being defined when it is different than the name of the enclosing port. A **clockDriver** contains all the following elements.

a) **clockPeriod** (type: ***realExpression*** (see C.3.1)) specifies the overall length (in time) of one cycle of the waveform. This element also contains a **units** (optional) attribute for specifying the units of the time values: **ns** (the default) and **ps**, where **ns** stands for nanosecond and **ps** stands for picosecond.

b) **clockPulseOffset** (type: ***realExpression*** (see C.3.1)) specifies the time delay from the start of the waveform to the first transition. This element also contains a **units** (optional) attribute for specifying the units of the time values: **ns** (the default) and **ps**, where **ns** stands for nanosecond and **ps** stands for picosecond.

c) **clockPulseValue** (type: ***unsignedBitVectorExpression*** (see C.3.6)) specifies the logic value to which the port transitions. This value is also the opposite of the value from which the waveform starts. The value of this element shall be `0` or `1`.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

d) **clockPulseDuration** (type: ***realExpression*** (see C.3.1)) specifies how long the waveform remains at the value specified by **clockPulseValue**. This element also contains a **units** (optional) attribute for specifying the units of the time values: **ns** (the default) and **ps**, where **ns** stands for nanosecond and **ps** stands for picosecond.

See also: SCR 12.7 and SCR 12.9.

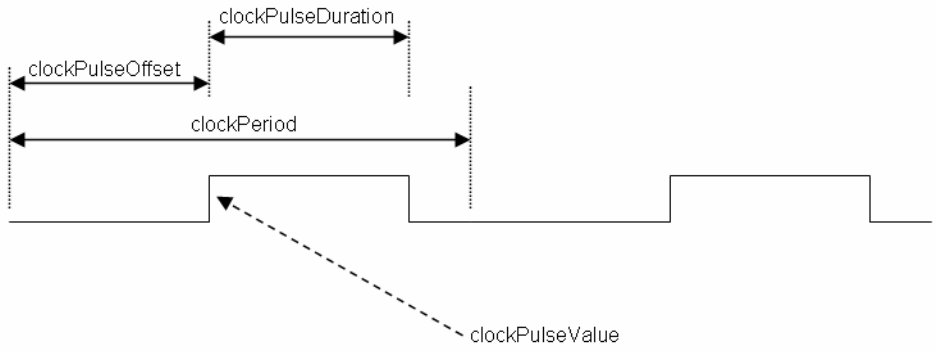


Figure 11—clockDriver elements

6.12.12 Implementation constraints

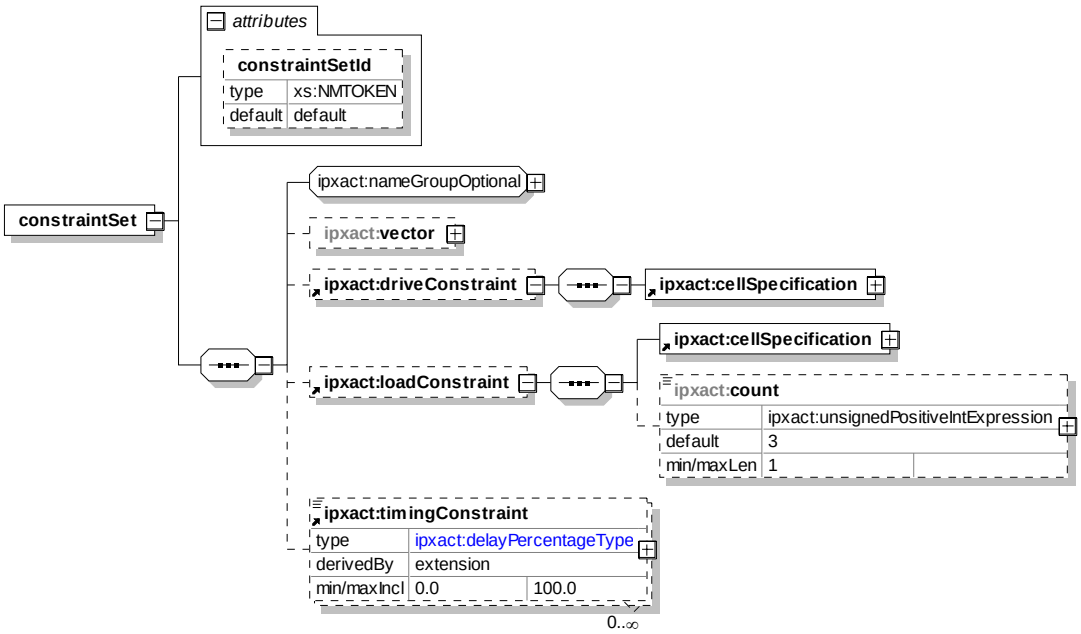
Implementation constraints can be defined to document requirements that need to be met by an implementation of the component. Constraints are defined in groups called *constraint sets* (in the IP-XACT element **port**/**wire**/**constraintSets**/**constraintSet**) so different constraints can be associated with different views of the component. A particular set of constraints is tied to a component view by the **constraintSetId** attribute in the constraint set and the matching **constraintSetRef** element in the view.

6.12.13 Component wire port constraints

6.12.13.1 Schema

The following schema defines the information contained in the **constraintSets** element, which may appear within a **wire** element within a component **port** element (**component**/**model**/**ports**/**port**/**wire**).

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



6.12.13.2 Description

The **constraintSets** element is used to define technology independent implementation constraints associated with the containing wire port of the component. The **constraintSets** element contains one or more **constraintSet** elements that define a set of constraints for the port. If more than one **constraintSet** element is present, each shall have a unique value for the **constraintSetId** (mandatory; type: ***NMTOKEN***; default: **default**) attribute so each **constraintSet** can be uniquely referenced from a **view**. **constraintSet** contains the following elements.

a) ***nameGroupOptional*** is defined in C.14.

b) **vector** (optional) determines to which bits of a vectored port the constraint applies. The **left** and **right** vector bounds elements inside the **vector** element specify the bounds of the vector. The **left** and **right** elements are of type ***nonNegativeInteger***.

c) **driveConstraint** (optional) defines a drive constraint for this port. The **driveConstraint** element may appear within a **modeConstraints** or **mirroredModeConstraints** element within a wire type port in an abstraction definition or within a **constraintSet** element within a wire type port in a component.

   The **driveConstraint** element defines a technology-independent drive constraint associated with the containing wire port of a component or the component port associated with the logical port within an abstraction definition if the **driveConstraint** element is contained within an abstraction definition. The actual constraint consists of a technology-independent specification of a library cell presumed to drive the input port. The **cellSpecification** element defines the cell (see 6.12.15).

   The **driveConstraint** element is not valid on an output port.

d) **loadConstraint** (optional) defines a load constraint for this port. The **loadConstraint** element may appear within a **modeConstraints** or **mirroredModeConstraints** element within a wire type port in an abstraction definition or within a **constraintSet** element within a wire type port in a component.

   The **loadConstraint** element defines a technology-independent load constraint associated with the containing wire port of a component or the component port associated with the logical port within an abstraction definition if the **loadConstraint** element is contained within an abstraction definition.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

The actual constraint consists of two parts, the technology-independent specification of a library cell and a count. **loadConstraint** also contains the following elements.

1) **cellSpecification** (mandatory) defines the library cell (see 6.12.15).
2) **count** (optional; type: ***unsignedPositiveIntExpression*** (see C.3.9); default: **3**) indicates how many loads of the indicated type are modeled as if attached to the output port.

The **loadConstraint** element is not valid on input ports.

e) **timingConstraint** (optional) defines a timing constraint relative to a clock for this port. See 6.12.14.

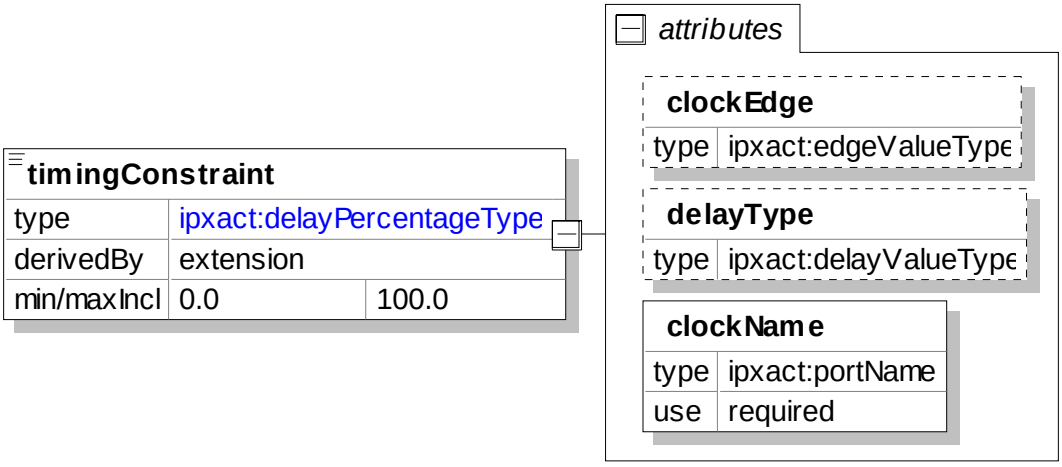
See also: SCR 12.1, SCR 12.2, SCR 12.3, SCR 12.4, SCR 12.5, and SCR 12.6.

NOTE—To specify technology-dependent constraints (which is not represented in the schema), use an SDC file and reference the file via **fileSet**.

6.12.14 Port timing constraints

6.12.14.1 Schema

The following schema defines the information contained in the **timingConstraint** element, which may appear within a **modeConstraints** or **mirroredModeConstraints** element within a wire type port in an abstraction definition or within a **constraintSet** element within a wire type port in a component.



6.12.14.2 Description

The **timingConstraint** element defines a technology-independent timing constraint associated with the containing wire port of a component or abstraction definition. It is of type ***delayPercentageType***; the value is a floating point number between `0` and `100`, which represents the percentage of the cycle time to be allocated to the timing constraint on the port. If the component port is an input (or the port in an abstraction definition ends up mapping to a physical port with direction **in**), the timing constraint represents an input delay constraint; otherwise, it represents an output delay constraint. **timingConstraint** also contains the following attributes.

a) **clockEdge** (optional; default: **rise**) specifies to which edge of the clock the constraint is relative. The default behavior is that the constraint is relative to the rising edge of the clock. The **clockEdge** attribute may have two values **rise** or **fall**.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

b) **delayType** (optional) restricts the constraint to applying to only best-case (minimum) or worst-case (maximum) timing analysis. By default, the constraint is applied to both. The **delayType** attribute may have two values **min** or **max**.

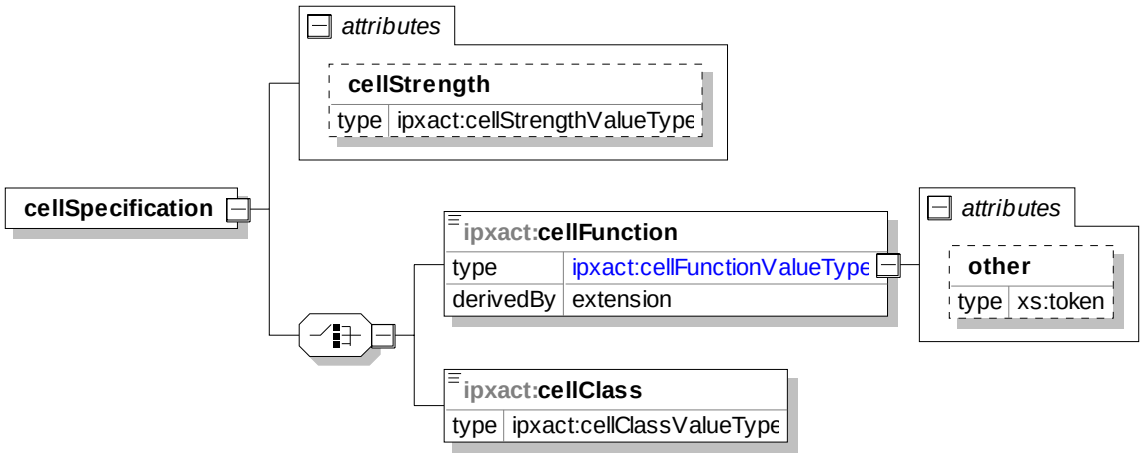
c) **clockName** (mandatory; type: ***portName***) specifies the delay constraint relative to the clock. **clockName** shall be a valid port name or another clock name in the containing description. The cycle time of the referenced clock is what actually determines the actual magnitude of the delay constraint (<clock cycle time> × 100 / <timing constraint element value>).

See also: SCR 12.7 and SCR 12.8.

6.12.15 Load and drive constraint cell specification

6.12.15.1 Schema

The following schema defines the information contained in the **cellSpecification** element, which may appear within a **loadConstraint** or **driveConstraint** element indicating the type of cell to use in the constraint.



6.12.15.2 Description

The **cellSpecification** element defines a cell in a technology-independent fashion such that drive and load constraints can be defined without referencing a specific technology library. The cell is defined so a DE can map it to an appropriate cell in a specific library when the actual constraint is generated.

a) **cellStrength** (optional) specifies the desired strength for this cell.

b) The **cellSpecification** element shall contain one of the following elements.

1) **cellFunction** specifies a cell function from the user-defined library. The **cellFunction** element shall be one of the following values: **nd2**, **buf**, **inv**, **mux21**, **dff**, **latch**, **xor2**, or **other**.

i) The **cellFunction** element contains a **cellStrength** (optional; default: **median**) attribute that provides the cell strength specification. The value shall be one of **low**, **median**, or **high**. **median** implies the middle cell of all the cells that match the desired function, sorted by drive or load strength (as appropriate for the given constraint), is used.

ii) When the value `other` is used, the **other** attribute (type: ***token***) can be set to a user-defined cell function. This attribute should only be used for documentation purposes as user-defined cell values are not guaranteed to be interpreted by DEs.

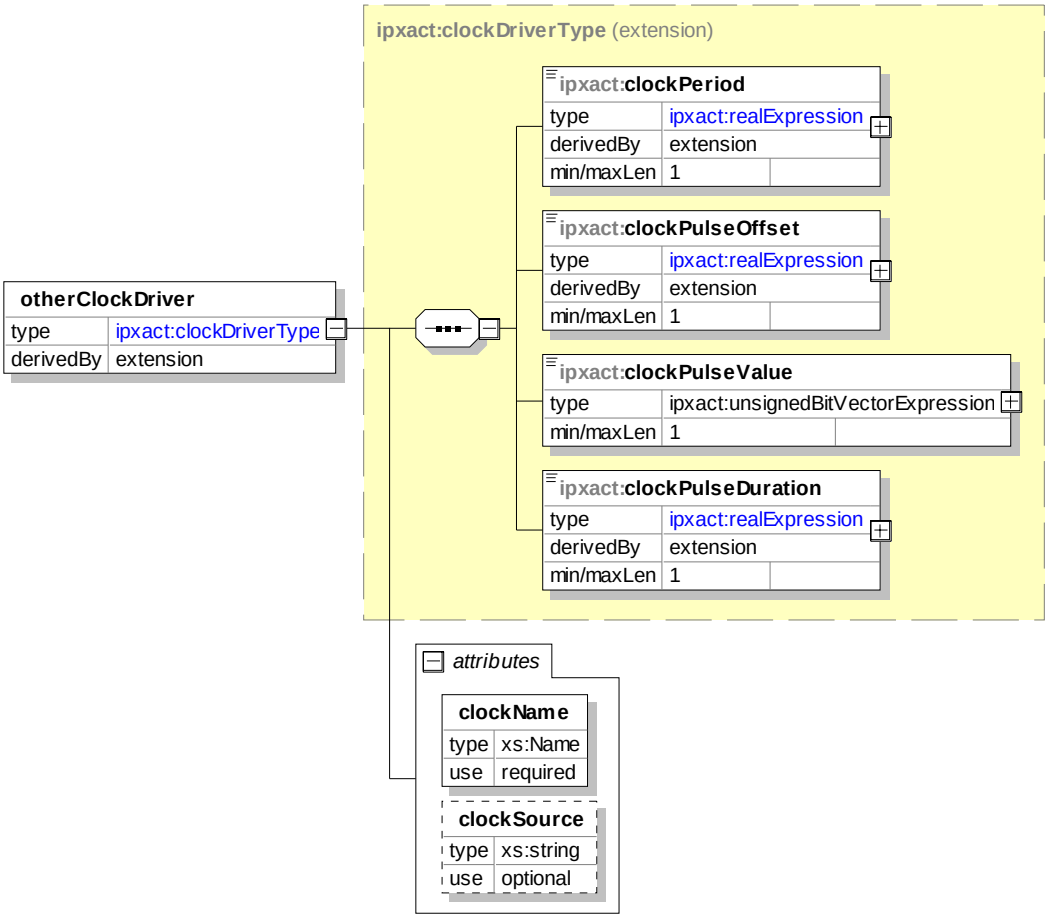
Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

2) **cellClass** specifies a cell class from the user-defined library. The **cellClass** element shall be one of the following values: **combinational** or **sequential**. The **cellClass** element contains a **cellStrength** (optional; default: **median**) attribute that provides the cell strength specification. The value shall be one of **low**, **median**, or **high**. **median** implies the middle cell of all the cells that match the desired class, sorted by drive or load strength (as appropriate for the given constraint), is used.

6.12.16 Other clock drivers

6.12.16.1 Schema

The following schema defines the information contained in the **otherClockDrivers** element, which may appear within a **component** element.



6.12.16.2 Description

The **otherClockDrivers** element defines clocks within a component that are not directly associated with a top-level port, e.g., virtual clocks and generated clocks. The **otherClockDrivers** element contains one or more **otherClockDriver** elements, each of which represents a single clock. The **otherClockDriver** element consists of a number of subelements that define the format of the clock waveform.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

a) **clockPeriod**, **clockPulseOffset**, **clockPulseValue**, and **clockPulseDuration** (all required) are all detailed in the description of the element **clockDriver**. See 6.12.11.

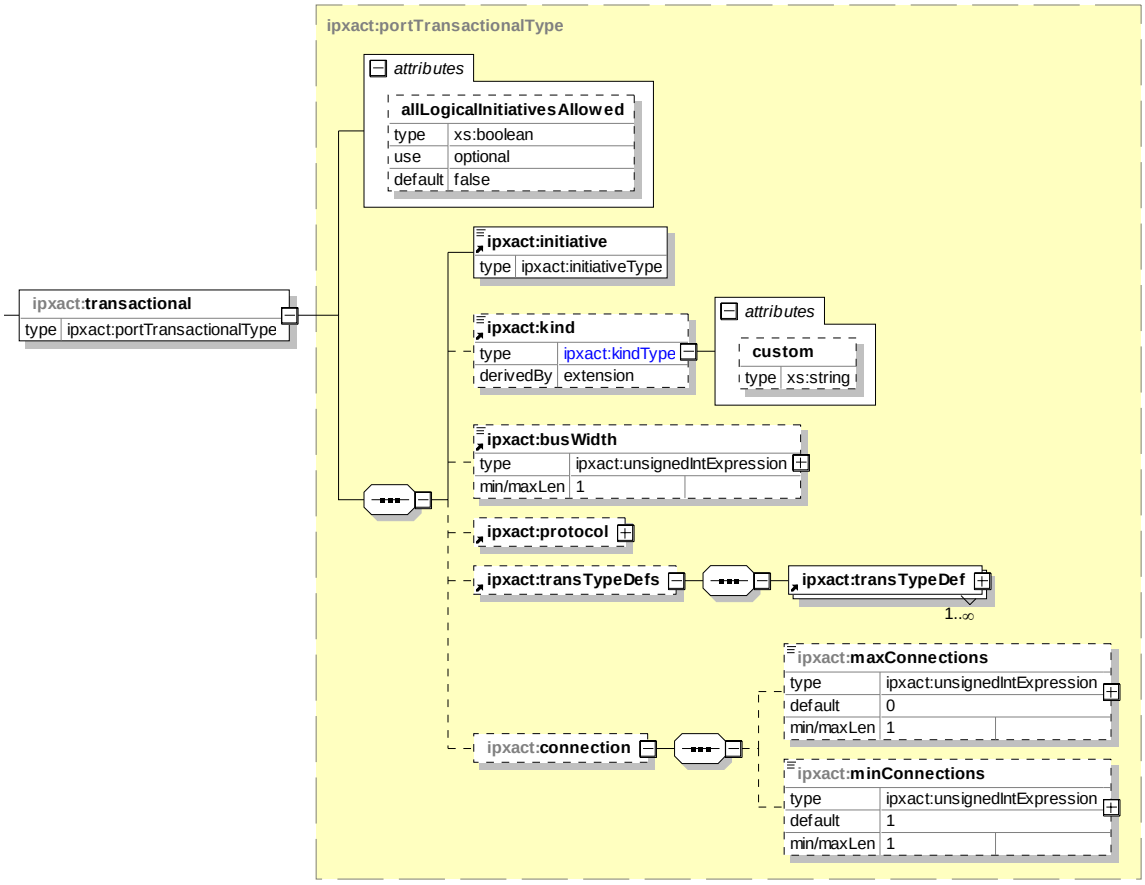
b) **clockName** (mandatory; type: ***Name***) attribute indicating the name of the clock for reference by a constraint.

c) **clockSource** (optional; type: ***string***) attribute defines the physical path and name of the clock generation cell.

6.12.17 Component transactional port type

6.12.17.1 Schema

The following schema defines the information contained in the **transactional** element (in a **component**/**model**/**ports**/**port** element).



6.12.17.2 Description

A **transactional** element in a component model port defines a physical transactional port of the component, which implements or uses a service. A service can be implemented with functions or methods. It contains the following elements.

a) **allLogicalInitiativesAllowed** (optional; type: ***boolean***; default: **false**) attribute defines whether the port may be mapped to a port in an **abstractionDefinition** with a different initiative. See 5.3.

b) **initiative** (mandatory) defines the type of access: **requires**, **provides**, **both**, or **phantom**.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

1) For example, a SystemC `sc_port` should be defined with the **requires** initiative, since it requires a SystemC interface. A SystemC `sc_export` should be defined with the **provides** initiative, since it provides a SystemC interface.

2) **both** indicates the type of access is both **requires** and **provides**.

3) **phantom** indicates a phantom port is being defined. See 6.12.21.

c) **kind** (optional) specifies the transactional port's type. It can take one of the following `enum` values:

tlm_port, **tlm_socket**, **simple_socket**, **multi_socket**, or **custom**. When **custom**, a *name* can be specified in the **custom** (type: ***string***) attribute. Also, the **tlm_port** should only be used for TLM1 notation; the other `enum` values should be used for TLM2 sockets.

d) **busWidth** (optional; type: ***unsignedIntExpression*** (see C.3.7)) specifies the data width in bits, e.g., `32` or `64`.

e) **protocol** (optional) characterizes how the information is transported by the transactional port. See 6.12.18.

f) **transTypeDefs** (optional) defines the port type expressed in the default language for this port. See 6.12.19.

g) **connection** (optional) defines the number of legal connections for a port.

1) **maxConnections** (optional; type: ***unsignedIntExpression*** (see C.3.7); default: **0**) indicating the maximum number of connections that this port supports. `0` indicates an unbounded number of legal connections.

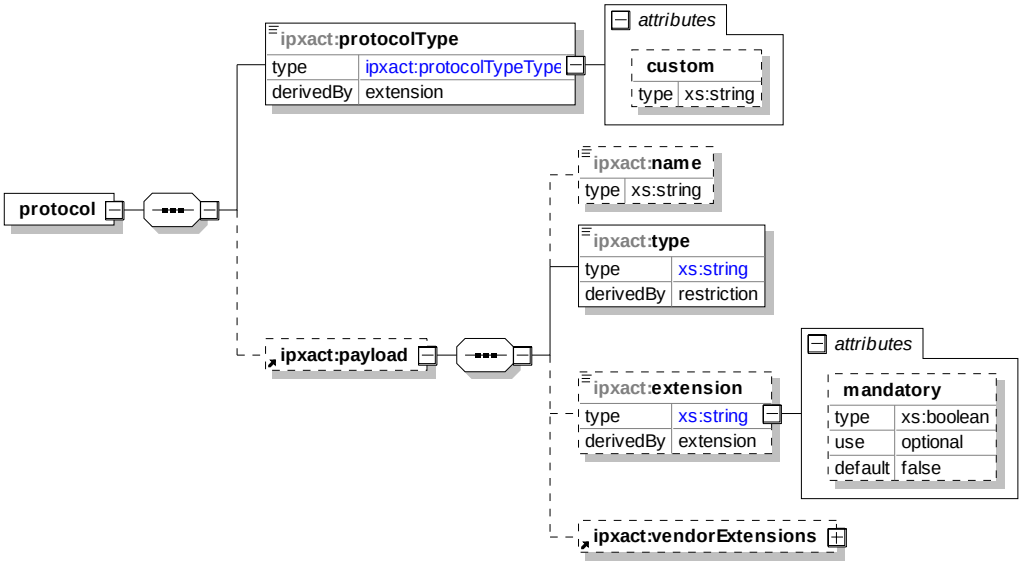
2) **minConnections** (optional; type: ***unsignedIntExpression*** (see C.3.7); default: **1**) indicating the minimum number of connections that this ports supports.

See also: SCR 6.2, SCR 6.3, SCR 6.4, SCR 6.13, SCR 6.22, and SCR 6.23.

6.12.18 Component transactional protocol/payload definition

6.12.18.1 Schema

The following schema defines the information contained in the **protocol**/**payload** element (in a **component**/**model**/**ports**/**port**/**transactional** element).



Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

6.12.18.2 Description

A **protocol** element characterizes the communication protocol. It contains a **protocolType** and (optionally) a **payload**. The **protocolType** can be **tlm** or **custom**.

A **payload** element specifies how the information is transported It contains the following subelements.

a) **name** (optional; type: ***string***) specifies the payload's name.

b) **type** (mandatory; type: ***string***) specifies if this is a **generic** or **specific** typing.

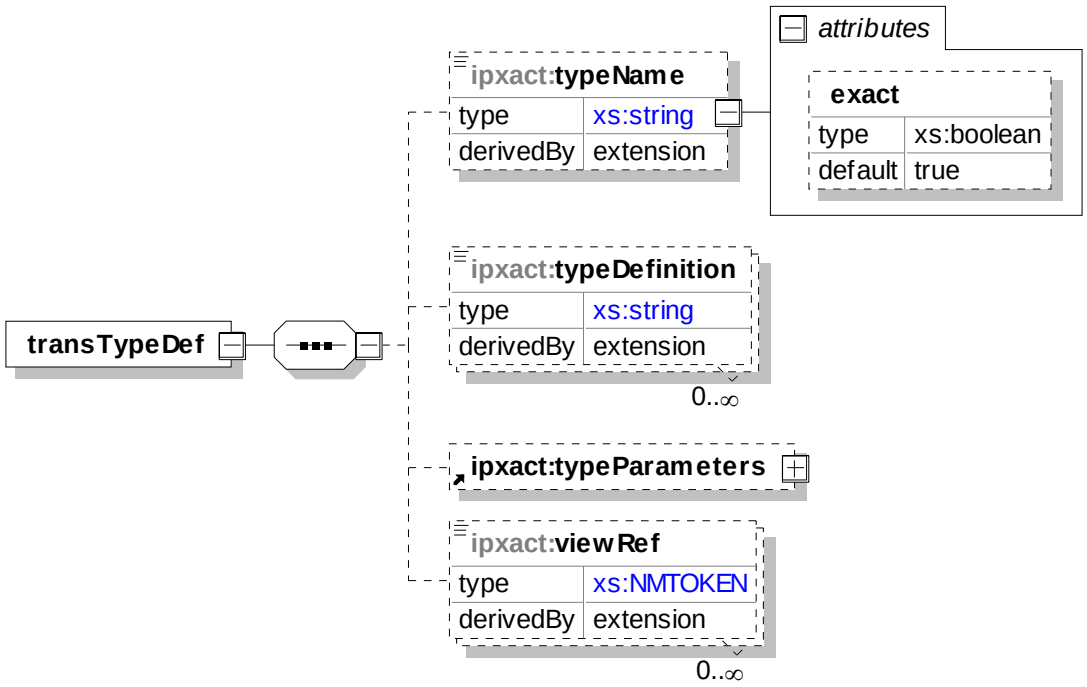
c) **extension** (optional; type: ***string***; default: **optional**) determines if this payload extension is **mandatory** or **optional**.

d) **vendorExtensions** (optional) adds any extra vendor-specific data related to the service. See C.24.

6.12.19 Component transactional port type definition

6.12.19.1 Schema

The following schema defines the information contained in the **transTypeDef** element (in a **component**/**model**/**ports**/**port**/**transactional** element).



6.12.19.2 Description

A **transTypeDef** element defines the port type expressed in the default language for this port (e.g., SystemC or SystemVerilog). It contains the following elements.

a) **typeName** (mandatory; type: ***string***) defines the port type (such as `sc_port`/`sc_export` in SystemC or any user-defined type, such as `tlm_port`). The **typeName** element may be associated with an optional ***boolean*** **exact** attribute (default: **true**). If **false**, this indicates the port type is an abstract type that may not be related to an existing type in the language of the referenced view.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

b) **typeDefinition** (optional; type: ***string***) contains a language-specific reference to where the given type is actually defined. Table 5 shows some examples. There can be multiple **typeDefinition**s for each port.

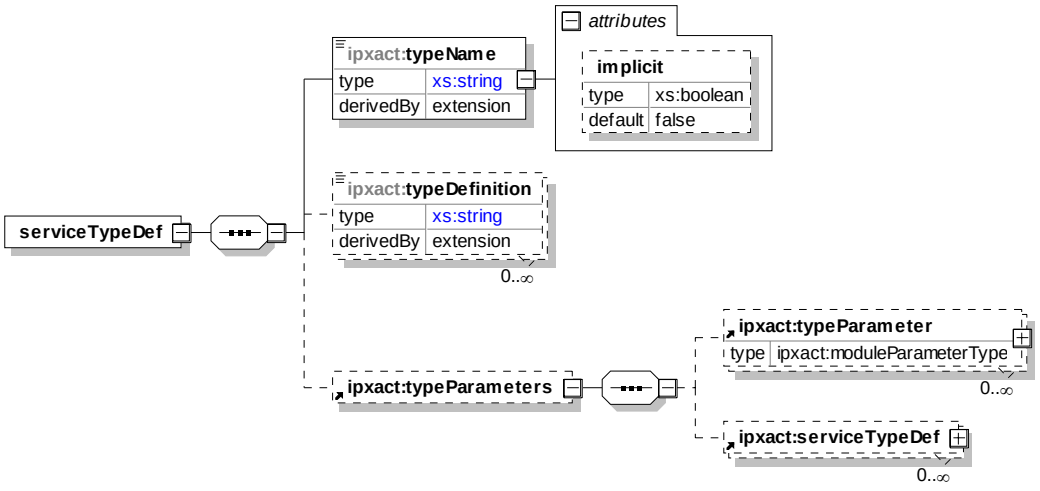
c) **typeParameters** (optional) specifies a list of port type parameters. See 6.12.20.

d) **viewRef** (optional; type: ***NMTOKEN***) specifies the views to which the **transTypeDef** applies. See C.26.

6.12.20 Component transactional port service

6.12.20.1 Schema

The following schema defines the information contained in the **serviceTypeDef** element (in a **component/model/ports/port/transactional/transTypeDefs/transTypeDef/typeParameters** element).



6.12.20.2 Description

A **serviceTypeDef** element describes the interface protocol associated with the transactional port. It contains the following elements and attributes.

a) **typeName** (mandatory; type: ***string***) defines the name of the service type (can be any predefined type, such as ***boolean*** or any user-defined type, such as `addr_type`). The **typeName** element may be associated with an ***boolean*** **implicit** attribute (default: **false**). If **true**, this indicates the service type should not be checked by SCRs.

b) **typeDefinition** (optional; type: ***string***) indicates a location where the type is defined, e.g., in SystemC and SystemVerilog; this is the include file containing the type definition.

c) **typeParameters** (optional) specifies any service type parameters. **typeParameter** is identical to **moduleParameter** (see 6.12.6). If a **typeParameter** describes the width of the transactional bus, the **typeParameter** element value shall match the **busWidth** element value in the transactional element (see 6.12.17.2).

6.12.21 Phantom ports

In some components, the RTL or TLM implementation of the component does not fully implement the functionality of the component described by IP-XACT. In RTL components, this is typically because the component has to work in design flows that only allow a signal to be routed though an RTL component if

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

there is some logic within the RTL component associated with that signal. This is particularly a problem for components containing channels or bridges.

An IP-XACT channel or bridge can represent the complete bus infrastructure between the master, slave, and system bus interfaces connected to the bus. As such, the component containing the channel should contain everything that is needed to create this infrastructure. In many buses, however, some signals are directly connected between the components attached to the bus, with no intervening logic. This is most often the case with clock and reset signals. If the component is to be usable in a wide range of design flows, these signals cannot be included in the RTL of the component.

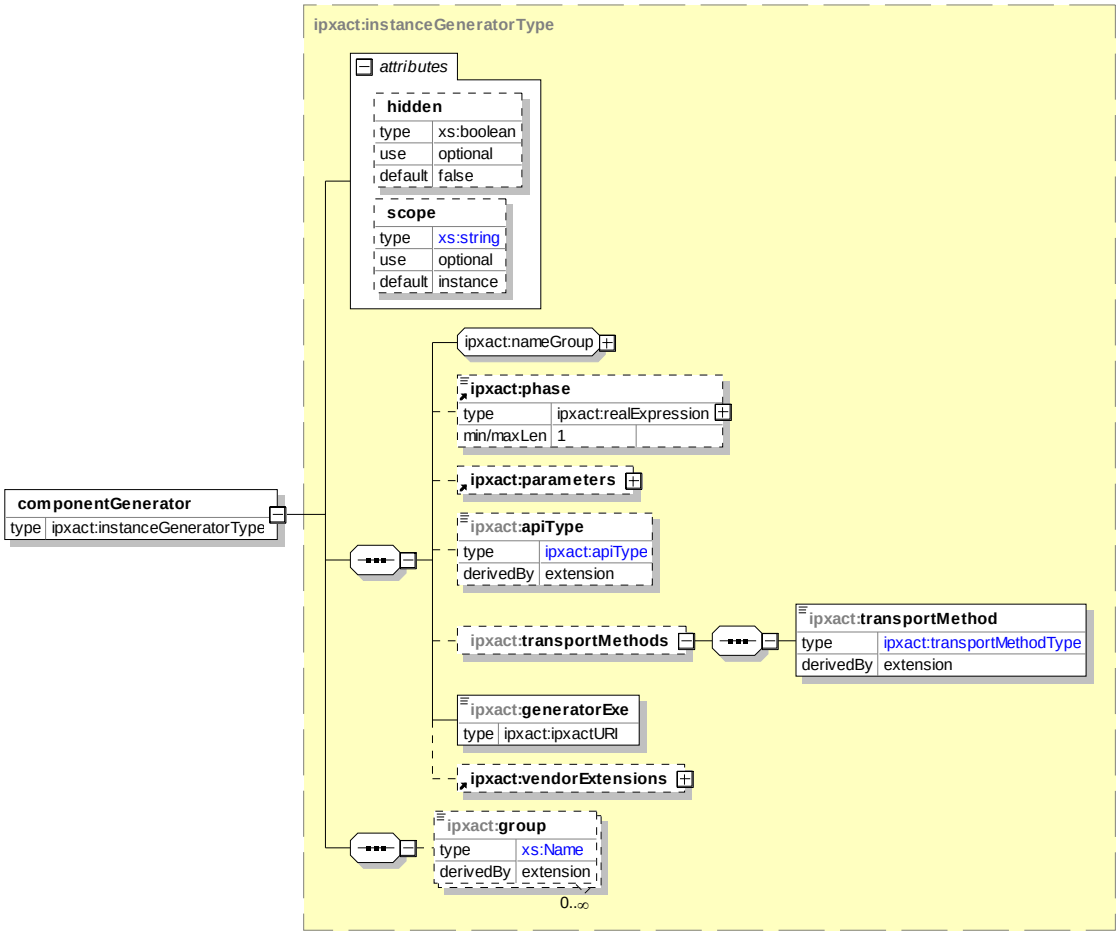
To fully describe such a channel or bridge component and allow netlisters that have no special knowledge of that bus type to netlist designs containing it, IP-XACT describes these additional connections as phantom ports. *Phantom ports* are additional ports included in the component's port list, but marked as **phantom**. As with real component ports, the mapping of a set of logical bus ports to that phantom port implies any design using that component needs to connect those logical ports with no intervening logic. The difference is a real component port needs to have a corresponding port in any RTL, TLM, or hierarchical IP-XACT implementation of the component; whereas, for phantom ports there is no corresponding port in the implementation.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

6.13 Component generators

6.13.1 Schema

The following schema details the information contained in the **componentGenerators** element, which may appear as an element inside the top-level **component** element.



6.13.2 Description

The **componentGenerators** element contains an unbounded list of **componentGenerator** elements. Each **componentGenerator** element defines a generator that is assigned and may be run on this component. **componentGenerator** contains the following attributes and elements.

a) The **hidden** (optional; type; ***boolean***; default: **false**) attribute specifies, when **true**, this generator shall not be run as a stand-alone generator and is required to be run as part of a chain. This generator should not be presented to the user for direct invocation. If **false**, this generator may be run as a stand-alone generator or in a generator chain.

b) The **scope** (optional; type; ***string***; default: **instance**) attribute is an enumerated list of **instance** and **entity**. **instance** indicates this generator shall be run once for all instances of this component. **entity** indicates this generator shall be run once for each instance of this component.

c) ***nameGroup*** group is defined in C.12. The **name** elements shall be unique within the containing **componentGenerators** element.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

d) **phase** (optional; type: ***realExpression*** (see C.3.1)) determines the sequence in which generators are run. Generators are run in order starting with zero (`0`). If two generators have the same phase number, the order shall be interpreted as not important and the generators can be run in any order. If no phase number is given, the generator is considered in the "last" phase and these generators are run in any order after the last generator with a phase number. The **phase** element also shall be a positive number.

e) **parameters** (optional) specifies any **componentGenerator** parameters. See C.18.

f) **apiType** (optional, type: **apiType**, default: **TGI_2014_BASE**) indicates the type of API used by the generator. Allowed values include:

1) **TGI_2014_BASE** indicates a generator which communicates using the base TGI API defined for this Standard.

2) **TGI_2014_EXTENDED** indicates a generator which communicates using the base and extended TGI APIs defined for this Standard.

3) **TGI_2009** indicates a generator which communicates using the TGI API defined for IEEE Std 1685-2009. This API is not part of this Standard and does not need to be supported by IP-XACT enabled DEs.

4) **none** indicates the generator does not communicate with the DE.

g) **transportMethods** (optional) defines alternate SOAP transport protocols that this generator can support. The default SOAP transport protocol is HTTP if this element is not present.

transportMethod specifies an alternate transport protocol. This element is an enumerated list of only one element **file**.

h) **generatorExe** (mandatory; type: ***ipxactURI***) contains an absolute or relative (to the location of the containing document) path to the generator executable. The path may also contain environment variables from the host system, which are used to abstract the location of the generator.

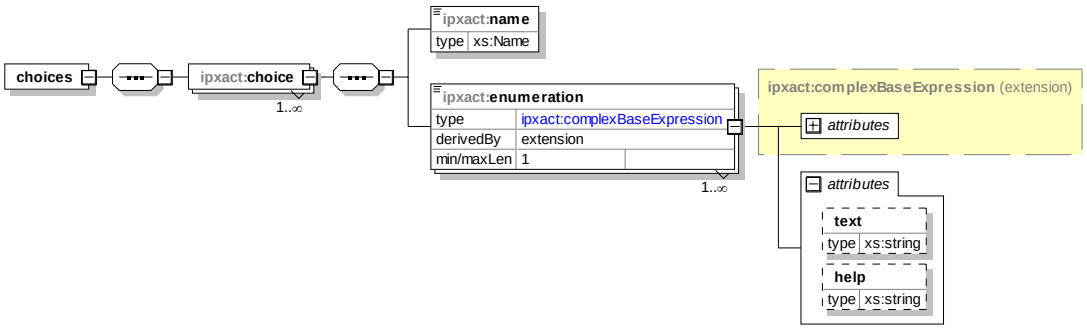
i) **vendorExtensions** (optional) adds any extra vendor-specific data related to the **componentGenerator**. See C.24.

j) **group** (optional; type: ***Name***) is an unbounded list of names used to assign this generator to a group with other generators. These **group** names are then referenced by a generator chain selector to forming a chain of generators. See 9.1.

6.14 Choices

6.14.1 Schema

The following schema details the information contained in the **choices** element, which may appear as an element inside the top-level **component**, **abstractor**, or **generatorChain** element.



Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

6.14.2 Description

The **choices** element contains an unbounded list of **choice** elements. Each **choice** element is a list of items used by a **moduleParameter** element, **parameter** element, or any other configurable element with a **choiceRef** attribute. These elements indicate they are using a **choice** element by setting the attribute **choiceRef**. This **choiceRef** attribute shall reference a valid **choice**/**name** element in the containing description.

The **choice** definition contains the following elements.

a) **name** (mandatory; type: ***Name***) specifies the name of this list and is used by other elements for reference. The **name** elements shall be unique within the containing **choices** element.

b) **enumeration** (mandatory; type: ***complexBaseExpression*** (see C.3)) is an unbounded list of elements, where each holds a possible value that the referencing element may contain.

   1) **text** (optional; type: ***string***) attribute causes optional text to be displayed when choosing the **choice** value. The resulting value stored in the configurable element corresponds to the enumeration value for the choice. If the **text** attribute is not present, the **enumeration** value may be displayed.

   2) **help** (optional; type: ***string***) attribute gives any additional information about this enumeration element.

See also: SCR 5.7.

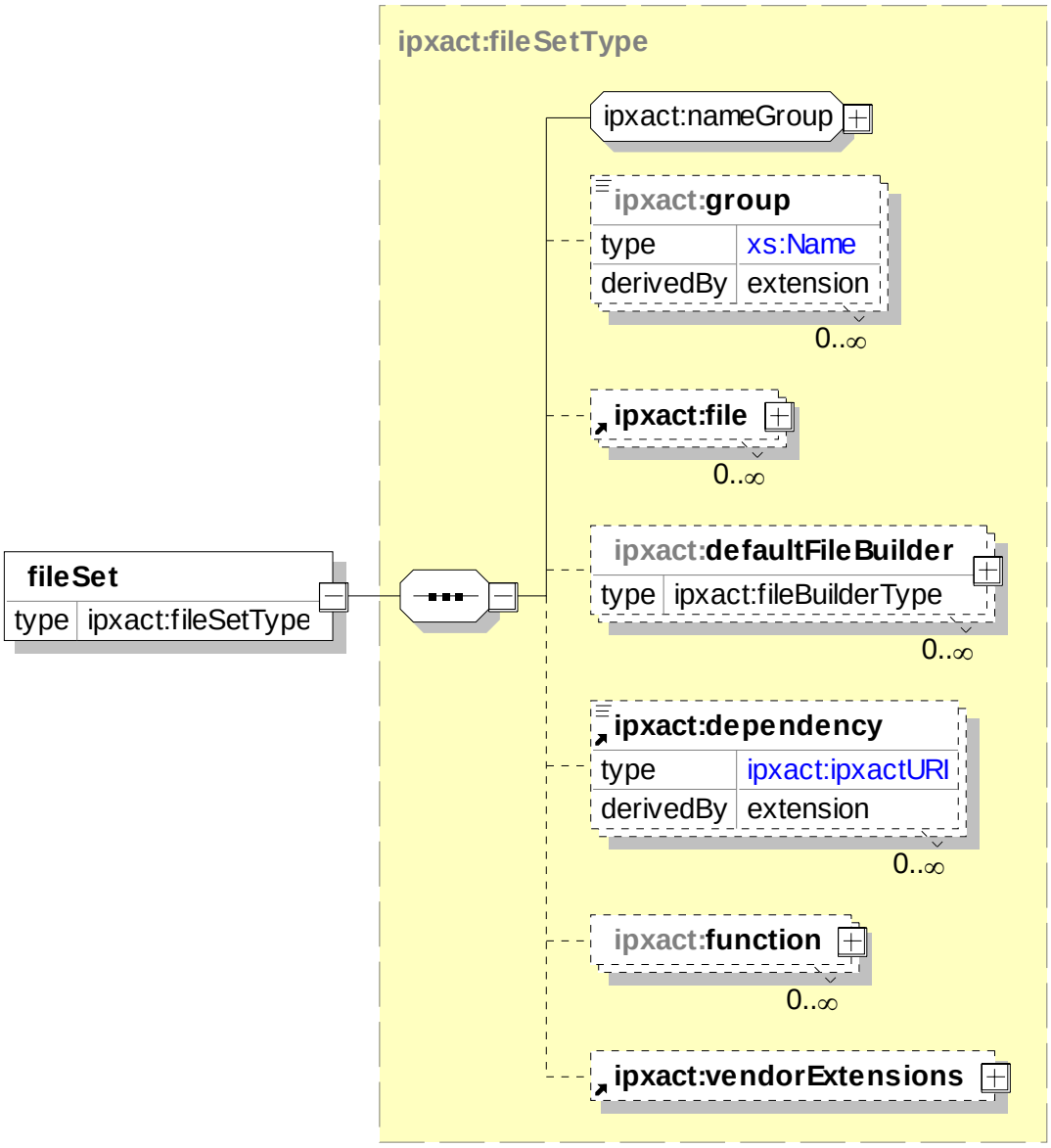
6.15 File sets

6.15.1 fileSets

6.15.1.1 Schema

The following schema details the information contained in the **fileSets** element, which may appear in a component or an abstractor.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



6.15.1.2 Description

The **fileSets** element contains one or more **fileSet** elements. A **fileSet** contains a list of files and directories associated with a component and/or instructions for further processing. If compilation order is important (e.g., for VHDL files), the files shall be listed in the order needed for compilation (the files to compile first are listed first). **fileSet** has the following mandatory and optional elements.

a) ***nameGroup*** group is defined in C.12. The **name** elements shall be unique within the containing **fileSets** element.

b) **group** (optional; type: ***Name***) describes the function or purpose of the file set with a single unbounded word group name (e.g., `diagnostics`, `interrupt`, etc.).

c) **file** (optional) references a single unbounded file or directory associated with the file set. If compilation order is important (e.g., for VHDL files), the files shall be listed in the order needed for compilation (see 6.15.2).

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

d) **defaultFileBuilder** (optional) specifies the unbounded default build commands for the files within this file set. See 6.15.4.

e) **dependency** (optional; type: ***ipxactURI***) is the path to a directory containing (include) files on which the file set depends.

f) **function** (optional) specifies the unbounded information about a software function for a generator (see 6.15.5).

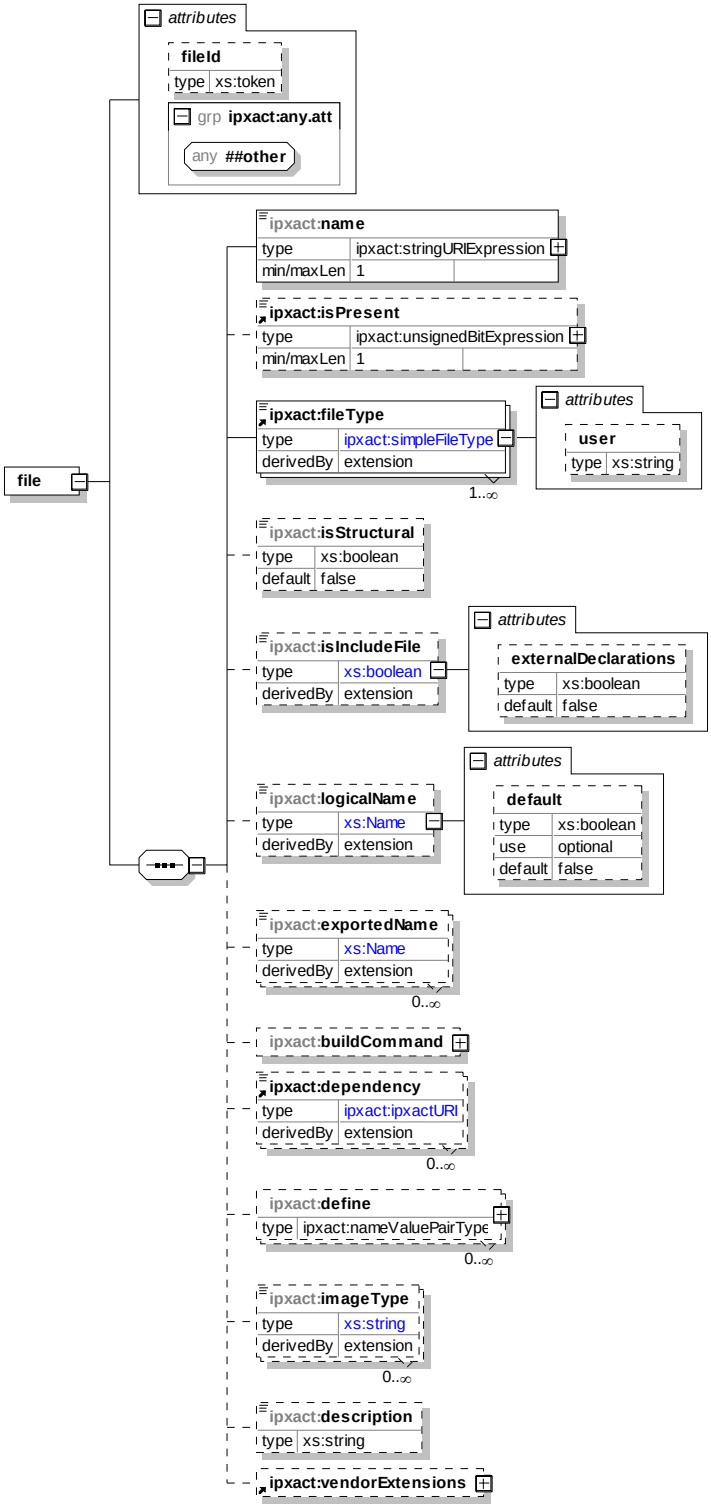
g) **vendorExtensions** (optional) provides a place for any vendor-specific extensions. See C.24.

6.15.2 file

6.15.2.1 Schema

The following schema details the information contained in the **file** element, which may appear as an element inside the **fileSet** element.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



6.15.2.2 Description

A **file** is a reference to a file or directory. It is an optional element of a **fileSet**. If compilation order is important (e.g., for VHDL files), the files shall be listed in the order needed for compilation (the files to

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

compile first are listed first). The **file** element contains an attribute **fileId** (optional; type: ***token***) that is used for references to this file from inside the **fileSet**/**function**/**fileRef** element. The **file** element also allows for vendor attributes to be applied. **file** contains the following elements.

a) **name** (mandatory; type: ***stringURIExpression*** (see C.3.4) contains an absolute or relative (to the location of the containing document) path to a file name or directory. The path may also contain environment variables from the host system in the form of `${ENV_VAR}`, used to abstract the location of files. See also D.11.

b) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

c) ***fileType*** (mandatory) group contains one or more of the elements defined in C.8.

d) **isStructural** (optional; type: ***boolean***; default: **false**) indicates this file contains only structural RTL. This is a content hint for any tools processing this file.

e) **includeFile** (optional; type: ***boolean***), when **true**, declares the file as an include file. If this element is not present the default value is **false**. **includeFile** has an attribute **externalDeclarations** (optional; type: ***boolean***; default: **false**); when **true**, this indicates the include file is needed by users of any files in this file set.

f) **logicalName** (optional; type: ***Name***) is the logical name for the file or directory, such as a VHDL library. **logicalName** includes an attribute **default** (optional; type: ***boolean***; default: **false**) that means (when **true**) the logical name shall only be used as a default and another process may override this name. If **false**, this logical name shall always be used.

g) **exportedName** (optional; type: ***Name***) defines any names that can be referenced externally.

h) **buildCommand** (optional) contains flags or commands for building the containing source file. These flags or commands override any flags or commands present in higher-level **defaultFileBuilder** elements. See 6.15.3.

i) **dependency** (optional; type: ***ipxactURI***) is the path to a directory containing (include) files on which the file depends.

j) **define** (optional; type: ***nameValuePairType*** (see C.17)) specifies the define symbols to use in the source file. This **define** element allows vendor attributes to be applied.

k) **imageType** (optional; type: ***string***) relates the current file to an executable image type in the design.

l) **description** (optional; type: ***string***) details the file for the user.

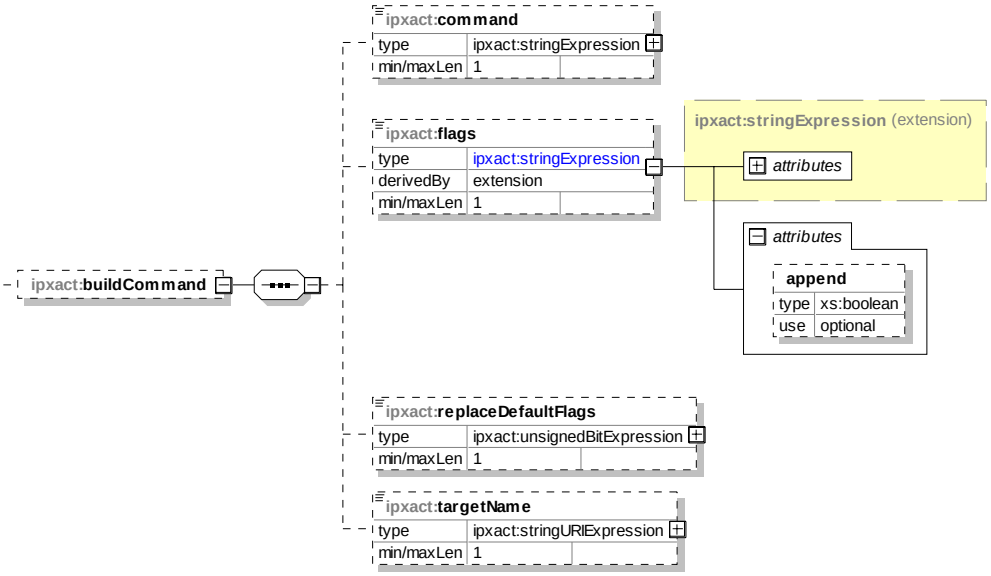
m) **vendorExtensions** (optional) provides a place for any vendor-specific extensions. See C.24.

6.15.3 buildCommand

6.15.3.1 Schema

The following schema details the information contained in the **buildCommand** element, which may appear as an element inside the **file** element.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



6.15.3.2 Description

A **buildCommand** contains flags or commands for building the containing source file. These flags or commands override any flags or commands present in higher-level **defaultFileBuilder** elements.

a) **command** (optional; type: ***stringExpression*** (see C.3.3)) element defines a compiler or assembler tool that processes files of this type.

b) **flags** (optional; type: ***stringExpression*** (see C.3.3)) documents any flags to be passed along with the software tool command. The **flags** element contains an attribute **append** (optional; type: ***boolean***), which when **true,** indicates the **flags** shall be appended to the current flags from the **defaultFile-Builder** (see 6.15.4), **fileBuilder** (see 6.8.5), or the build script generator. If **false**, the **flags** shall replace the existing flags.

c) **replaceDefaultFlags** (optional; type: ***unsignedBitExpression*** (see C.3.5.)), when **true**, documents flags that replace any of the default flags from the build script generator. If **false**, the flags are appended. If **true** and the **flags** element is empty or not present, this has the effect of clearing all the flags. If this element is not present, its effective value is **false**.

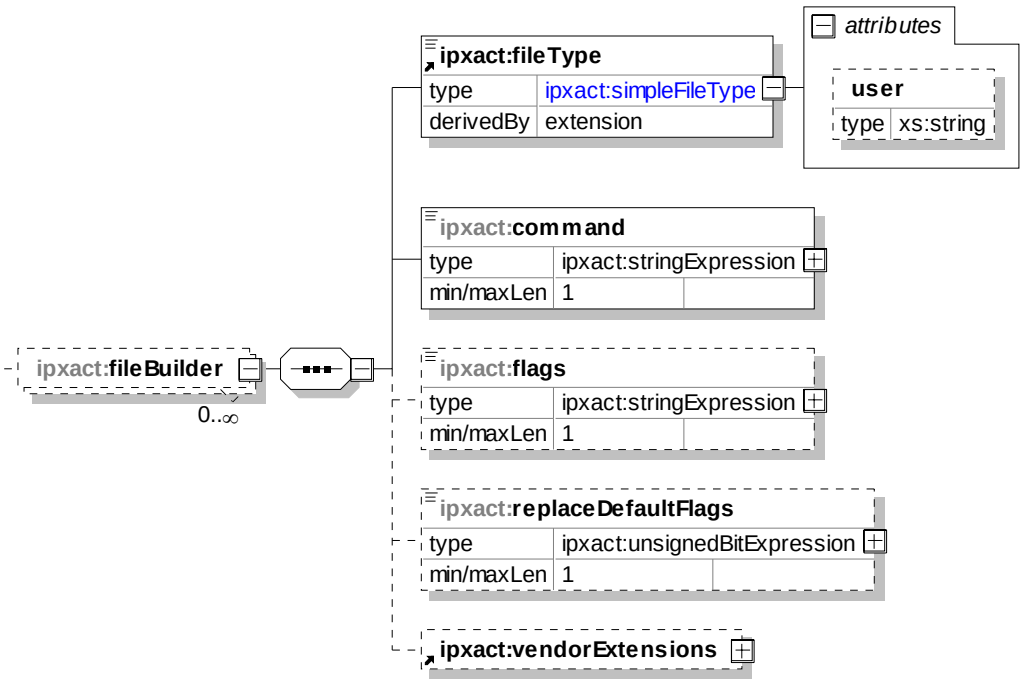
d) **targetName** (optional; type: ***stringURIExpression*** (see C.3.4)) defines the path to the file derived from the source file.

6.15.4 defaultFileBuilder

6.15.4.1 Schema

The following schema details the information contained in the **defaultFileBuilder** element, which may appear as an element inside the **fileSet** or **view** element.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



6.15.4.2 Description

A **defaultFileBuilder** contains default flags or commands for building the containing source file types. These flags or commands may be overridden by flags or commands present in lower-level **defaultFileBuilder** or **buildCommand** elements.

a) ***fileType*** (mandatory) group contains one or more of the elements defined in C.8. If the ***fileType*** is set to `other`, the optional attribute **other** (type: ***string***) can be set to indicate a user-defined file type. No semantic meaning is inferred from the user-defined file type.

b) **command** (optional; type: ***stringExpression*** (see C.3.3)) element defines a compiler or assembler tool that processes files of this type.

c) **flags** (optional; type: ***stringExpression*** (see C.3.3)) documents any flags to be passed along with the software tool command.

d) **replaceDefaultFlags** (optional; type: ***unsignedBitExpression*** (see C.3.5)) when **true** indicates the **flags** shall be appended to the current flags. If **false**, the **flags** shall replace the existing flags.

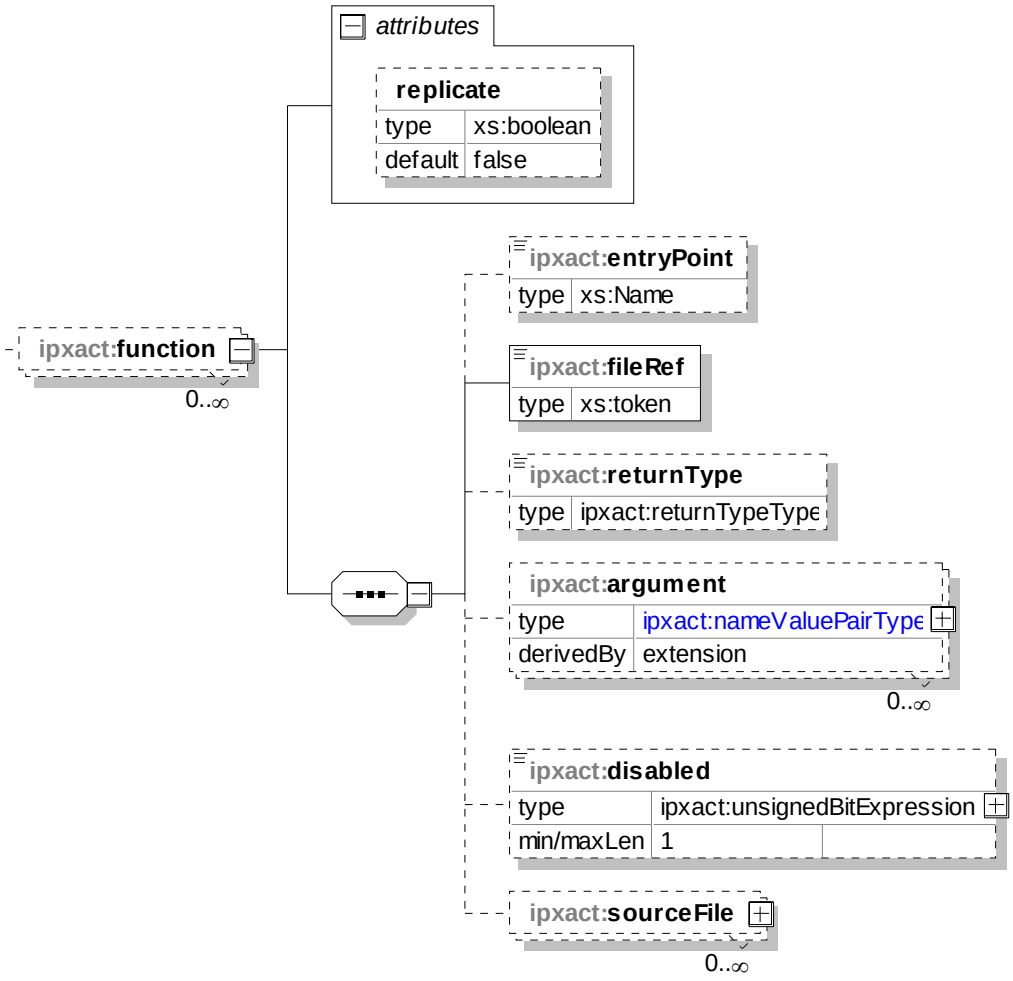
e) **vendorExtensions** (optional) provides a place for any vendor-specific extensions. See C.24.

6.15.5 function

6.15.5.1 Schema

The following schema details the information contained in the **function** element, which may appear as an element inside the **fileSet** element.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



6.15.5.2 Description

A **function** specifies information about a software function. **function** contains the **replicate** attribute (optional; type: ***boolean***; default: **false**); when set to **true**, the generator compiles a separate object module for each instance of the component in the design. This allows the function to be called with different attributes for each instance within the design (e.g., base address). **function** has the following elements.

a) **entryPoint** (optional; type: ***Name***) is the entry point name for the function or subroutine.

b) **fileRef** (mandatory type: ***token***) reference to the file that contains the entry point for the function. The value of this element shall match an attribute in **file**/**fileId**. See 6.15.2.

c) **returnType** (optional) is an enumerated ***string*** type that indicates the return type for the function. The two possible values are **int** and **void**.

d) **argument** (optional; type: ***nameValuePairType*** (see C.17)) specifies the arguments passed to the **function** when making a call. All arguments shall be passed in the order presented in this description. The **dataType** (mandatory) attribute specifies the type for this argument, e.g., an `int` or `boolean`. The **argument** element also allows for vendor attributes to be applied.

e) **disabled** (optional; type: ***unsignedBitExpression*** (see C.3.5)) disables the software function. When **true**, the software function is not available for use. When **false**, the function is available. If this element is not present, its effective value is **false**.

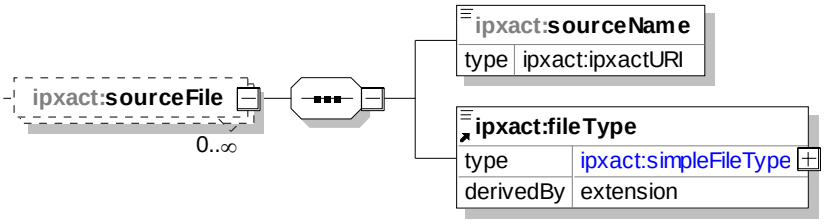
Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

f) **sourceFile** (optional) references any source files. The order of the source files may be important, as this could indicate a compile order. See 6.15.6.

6.15.6 sourceFile

6.15.6.1 Schema

The following schema details the information contained in the **sourceFile** element, which may appear as an element inside the **function** element.



6.15.6.2 Description

The **sourceFile** element specifies the location of the source files for this **function**. All source files shall be processed in the order presented in this description.

a) **sourceName** (mandatory; type: ***ipxactURI***) contains an absolute or relative (to the location of the containing document) path to a file name or directory. The path may also contain environment variables from the host system, used to abstract the location of files.

b) ***fileType*** (mandatory) group contains one or more of the elements defined in C.8.If the ***fileType*** is set to `other`, the optional attribute **other** can be set to indicate a user-defined file type. No semantic meaning is inferred from the user-defined file type.

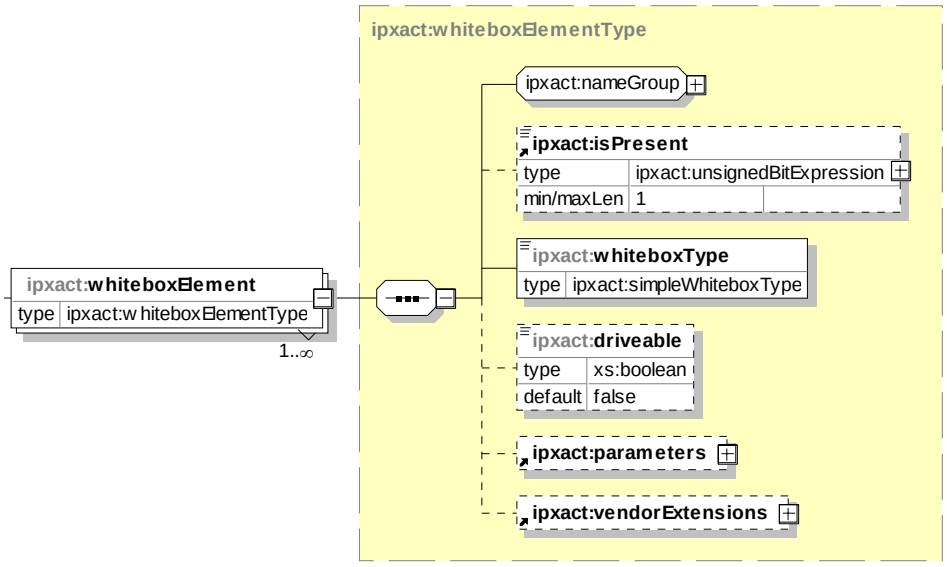
6.16 White box elements

Verification IP (VIP), with monitor bus interfaces, connect to an active bus interface to monitor only that interface's protocol for a variety of uses. Other verification tools may require access to component IP in a design, at a level deeper than the interfaces defined for the component. A white box element provides such access. This can be used in situations where internal registers, pins, signals, or whole IP-XACT interfaces need to be monitored or driven by VIP.

6.16.1 Schema

The following schema details the information contained in the **whiteboxElements** element, which may appear as an element inside the top-level **component** element.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



6.16.2 Description

The **whiteboxElements** element contains a list of one or more **whiteboxElement** elements. Each **whiteboxElement** element contains the following elements.

a) ***nameGroup*** group is defined in C.12. The **name** element shall be unique within the containing **whiteboxElements** element.

b) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

c) **whiteboxType** (mandatory) documents this white box element's referent: a **pin**, **signal**, or **interface** within the component. **pin** indicates a port on an internal instance in this component can be mapped to physical signals. **signal** indicates a signal between two internal instances in this component can be mapped to physical signals. **interface** indicates a group of signals that can be addressed as a single name.

   In each case, the view-specific path is contained in the matching **model/view/whiteboxElementRef** element.

d) **driveable** (optional; type: ***boolean***; default: **false**), when **true**, indicates the white box describes a point within the IP that can be driven, i.e., forced to a new value. If **false**, the white box references a point that cannot be driven. If this element is not present, its effective value is **false**.

e) **parameters** (optional) specifies any parameter names and types for a white box that can be parameterized. See C.18.

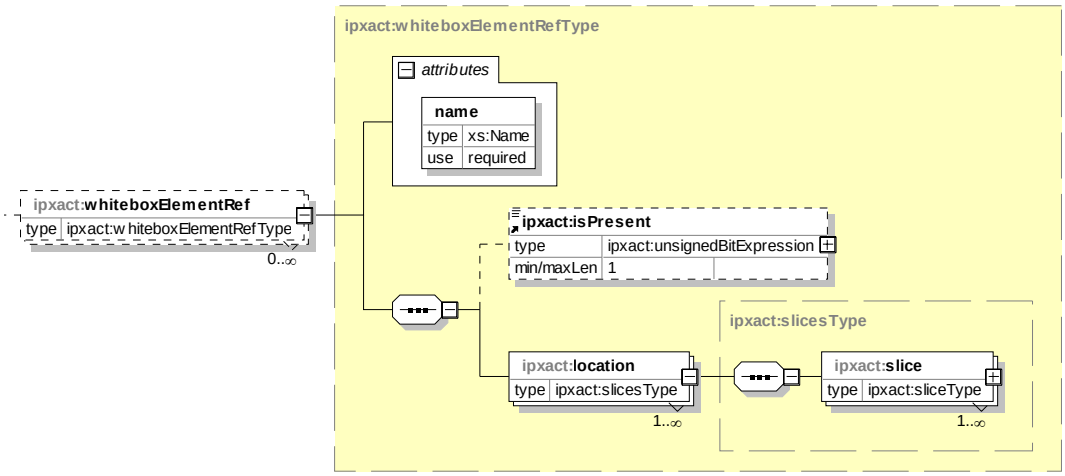
f) **vendorExtensions** (optional) provides a space for any vendor-specific extensions. See C.24.

6.17 White box element reference

6.17.1 Schema

The following schema details the information contained in the **whiteboxElementRefs** element, which may appear as an element inside the **component/model/instantiations/component** element.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



6.17.2 Description

The **whiteboxElementRefs** element contains a list of one or more **whiteboxElementRef** elements. The **whiteboxElementRef** makes a reference to a **whiteboxElement** of the component and defines the view specific path to the element. The **name** (mandatory; type: ***Name***) attribute identifies the **whiteboxElement** in the containing component for which the following **location** applies. **whiteboxElement** element contains the following elements.

a) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

b) **location** (mandatory) specifies the HDL specific path(s) to be associated with the indicated **whiteboxElement**. A **location** element defines a HDL path to the HDL representation of the IP-XACT object. If there are multiple **location** elements, the IP-XACT object has several copies in the HDL. Each **location** element contains one or more **slice** elements. (see C.1.5).

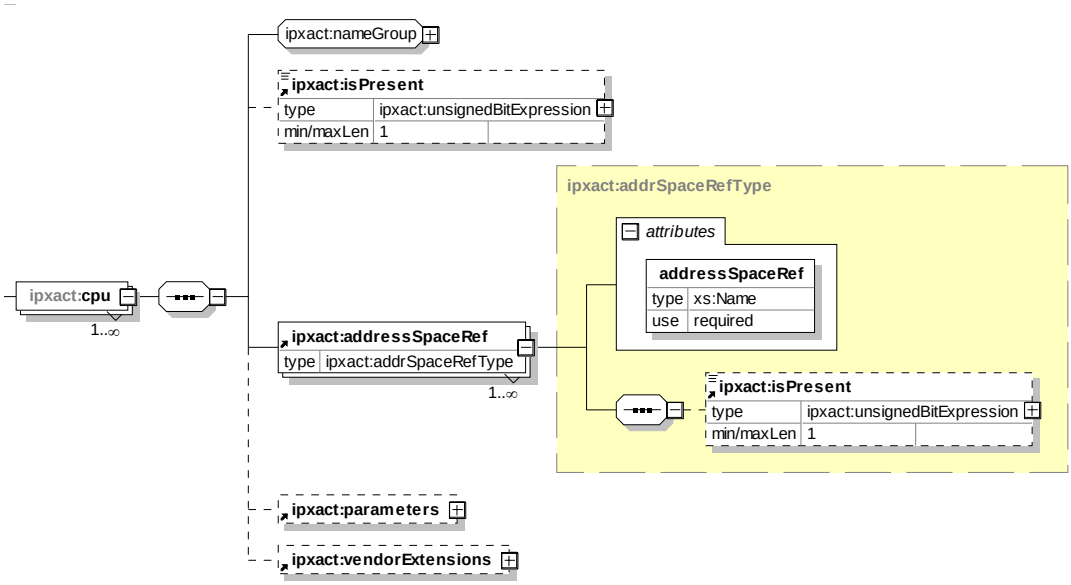
See also SCR 16.3.

6.18 CPUs

6.18.1 Schema

The following schema details the information contained in the **CPUs** element, which may appear as an element inside the top-level **component** element.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



6.18.2 Description

The **cpus** element contains an unbounded list of **cpu** elements for the containing component. The **cpu** element describes a containing component with a programmable core that has some sized address space. That same address space may also be referenced by a master interface and used to create a link for the programmable core to know from which interface transaction the software departs.

a) ***nameGroup*** group is defined in C.12. The **name** element shall be unique within the containing **component** element.

b) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

c) **addressSpaceRef** (mandatory) contains an attribute to describe information about the range of addresses with which the master interface related to this **cpu** can generate transactions.

   1) **addressSpaceRef** (mandatory; type: ***Name***) attribute references a name of an address space defined in the same component. The address space defines the range and width for transaction on this interface. See 6.8.1.

   2) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

d) **parameters** (optional) specifies any **cpu**-type parameters. See C.18.

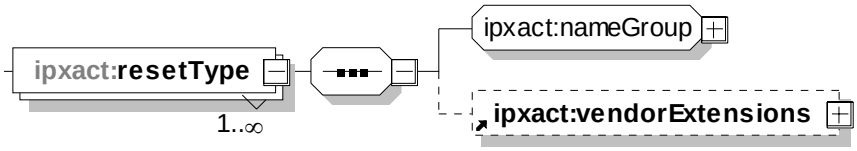
e) **vendorExtensions** (optional) adds any extra vendor-specific data related to the **cpu**. See C.24.

6.19 Reset types

6.19.1 Schema

The following schema details the information contained in the **resetTypes** element, which may appear as an element inside the top-level **component** element.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



6.19.2 Description

The **resetTypes** element contains an unbounded list of **resetType** elements for the containing component. The **resetType** element describes any user-defined reset types referenced within field reset elements. The reset type named **HARD** is the default and should be treated as being pre-defined. It is not legal to explicitly define a **resetType** element using that name. Register fields which do not reference a reset type are presumed to be of type HARD reset. A **resetType** contains the following elements.

a) ***nameGroup*** group is defined in C.12. The **name** element shall be unique within the containing **component** element.

b) **vendorExtensions** (optional) adds any extra vendor-specific data related to the **resetType**. See C.24.

See also: SCR 7.20.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

7. Design descriptions

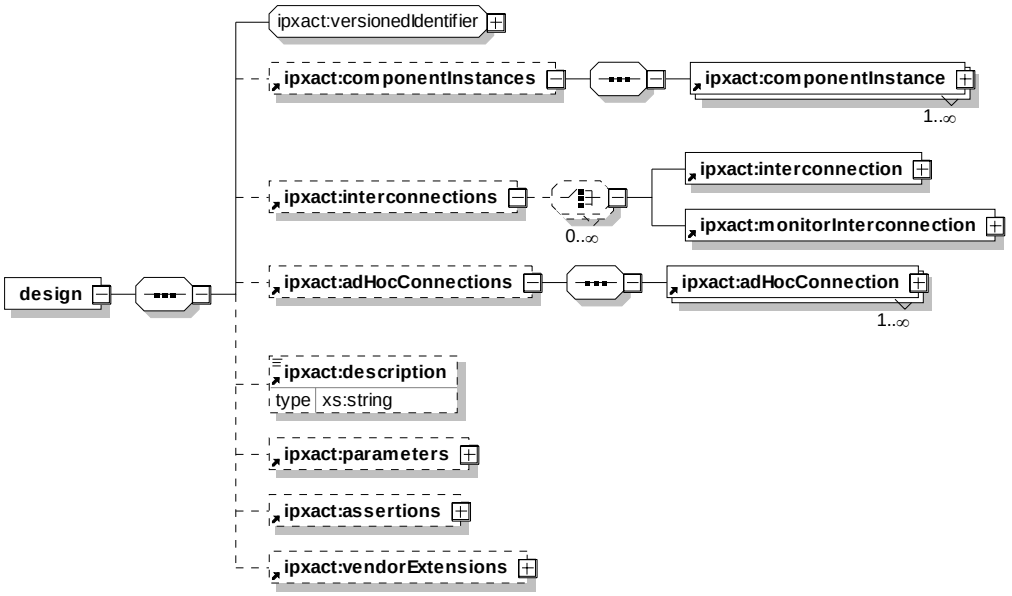
7.1 Design

An IP-XACT *design* is the central placeholder for the assembly of component objects meta-data. A design describes a list of components referenced by this description, their configuration, and their interconnections to each other. The interconnections may be between interfaces or between ports on a component. A design description is analogous to a schematic of components.

While a design description, with referenced components and interconnections, describes most of the information for a design, some information is missing, such as the exact port names used by a bus interface. To resolve this a component description (referred to as a *hierarchical component*) is used. This *component description* contains a view with a reference to the design description. Together, the component and referenced design description form a complete single-level hierarchical description. From this point, it is simple to create additional hierarchical descriptions by including hierarchical component description in design descriptions.

7.1.1 Schema

The following schema details the information contained in the **design** element, which is one of the top-level elements of the schema.



7.1.2 Description

The **design** element describes a list of referenced components, their configuration and interconnections to each other. Each element of a **design** is detailed in the rest of this clause; the main sections of a **design** are:

a) ***versionedIdentifier*** group provides a unique identifier, made up of four subelements for a top level IP-XACT element. See C.25.

b) **componentInstances** (optional) contains an unbounded list of **componentInstance** elements, documenting components that are instantiated (referenced) inside the design (see 7.2).

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

c) **interconnections** (optional) contains the connections between bus interfaces of components listed inside the design. The **interconnection** element is used to document 'active' and hierarchical interface connections and the **monitorInterconnection** element is used to document a connection between an active interface and a monitor interface (see 7.3).

d) **adHocConnections** (optional) contains an unbounded list of **adHocConnection** elements, documenting connections between component ports listed inside this design (see 7.5).

e) **description** (optional) allows a textual description of the design. The **description** element is of type ***string***.

f) **parameters** (optional) describes any **parameter** that can be used to configure or hold information related to this **design**. See C.18.

g) **assertions** (optional) contains a list of expressions defining the allowed parameter values. See C.2.

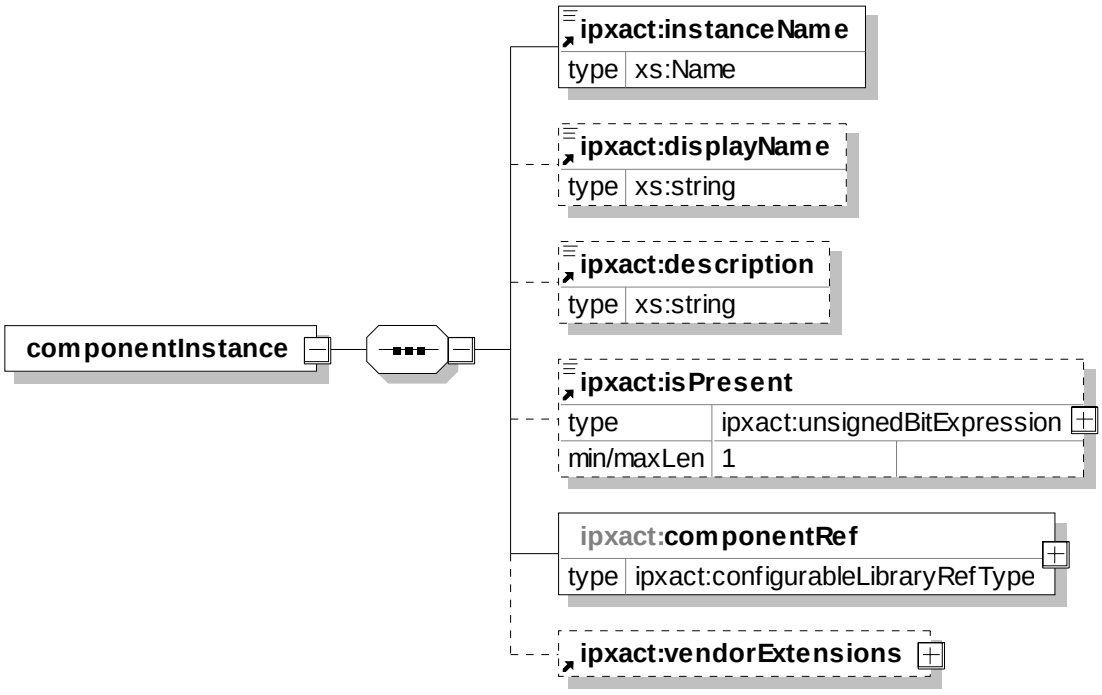
h) **vendorExtensions** (optional) adds any extra vendor-specific data related to the design. See C.24.

See also: SCR 1.10.

7.2 Design component instances

7.2.1 Schema

The following schema details the information contained in the **componentInstances** element, which may appear as an element inside the top-level **design** element.



7.2.2 Description

The **componentInstance** element documents the existence of a component in a design. This element contains the following subelements.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

a) **instanceName** (mandatory; type: ***Name***) assigns a unique name for this instance of the component in this design. The value of this element shall be unique inside a **design** element.

b) **displayName** (optional type: ***string***) allows a short descriptive text to be associated with the instance.

c) **description** (optional type: ***string***) allows a textual description of the instance.

d) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

e) **componentRef** (mandatory; type ***configurableLibraryRefType*** (see C.6)) is a reference to a component description (see 6.1) for this component instance. The **componentRef** element contains four attributes to specify a unique VLNV and an optional **configurableElementValues** element, which is used to document values for parameters to be passed into the referenced document.

f) **vendorExtensions** (optional) adds any extra vendor-specific data related to the design. See C.24.

See also: SCR 1.9, SCR 5.3, SCR 5.12, and SCR 15.13.

7.3 Design interconnections

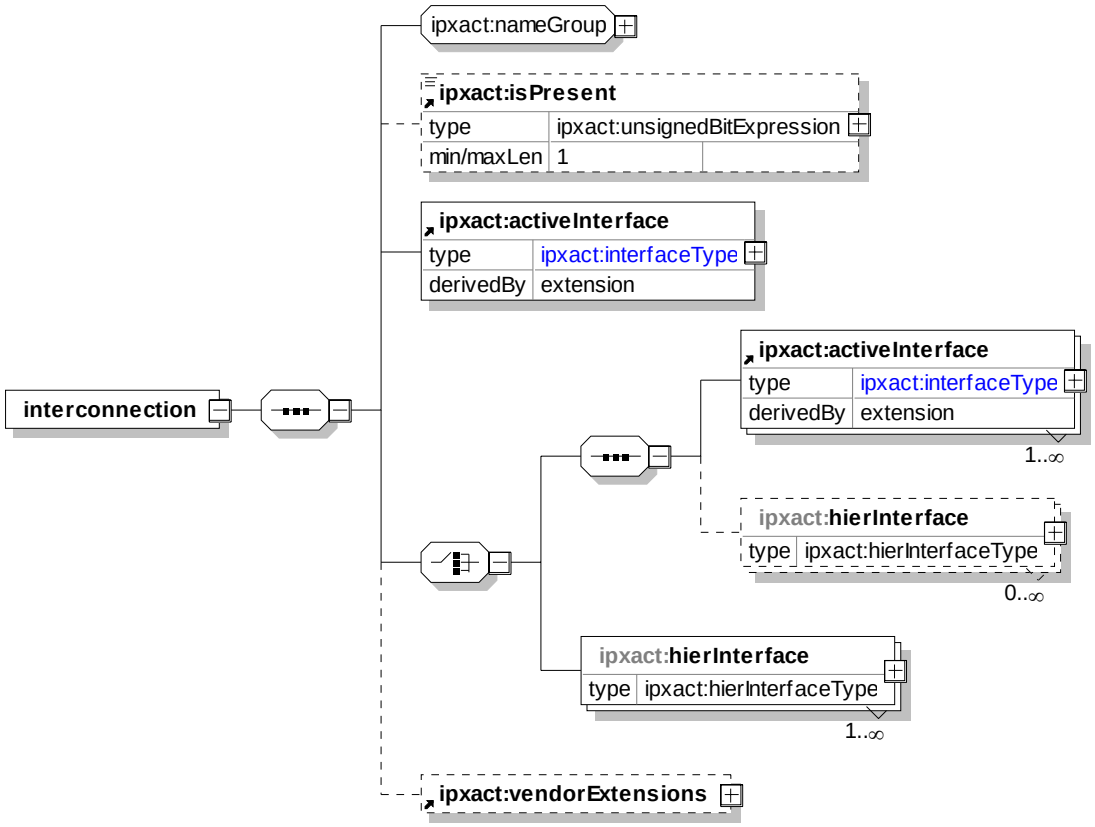
The **interconnections** element contains an unbounded list of **interconnection** (see 7.3.1) and **monitorInterconnection** (see 7.3.2) elements. The **interconnections** element may appear as an element inside the top-level **design** element. For further description on interface connections, see 6.3.5.

7.3.1 interconnection

7.3.1.1 Schema

The following schema details the information contained in the **interconnection** element, which may appear as an element inside the **interconnections** element.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



7.3.1.2 Description

The **interconnection** element specifies a connection between one bus interface of a component and another bus interface of a component. Each interconnection contain the following elements.

a) ***nameGroup*** group is defined in C.12. The **name** elements shall be unique within the containing **interconnections** element.

b) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

c) At least two interfaces need to be defined; the following combinations are possible here:
one **activeInterface** element and a choice of either
one or more additional **activeInterface** elements plus zero or more **hierInterface** elements, or
one or more **hierInterface** elements.

1) **activeInterface** (type: ***interfaceType*** (see 7.4)) specifies an 'active' (non-monitor) interface connection point on a component. Having more than two **activeInterface**/**hierInterface** elements present is only allowed under certain conditions.

2) **hierInterface** (type: ***hierInterfaceType*** (see 7.4)) specifies an ‘active’ (non-monitor) interface connection point on the boundary of the containing component. Having more than two **activeInterface**/**hierInterface** elements present is only allowed under certain conditions.

d) **vendorExtensions** (optional) adds any extra vendor-specific data related to the design. See C.24.

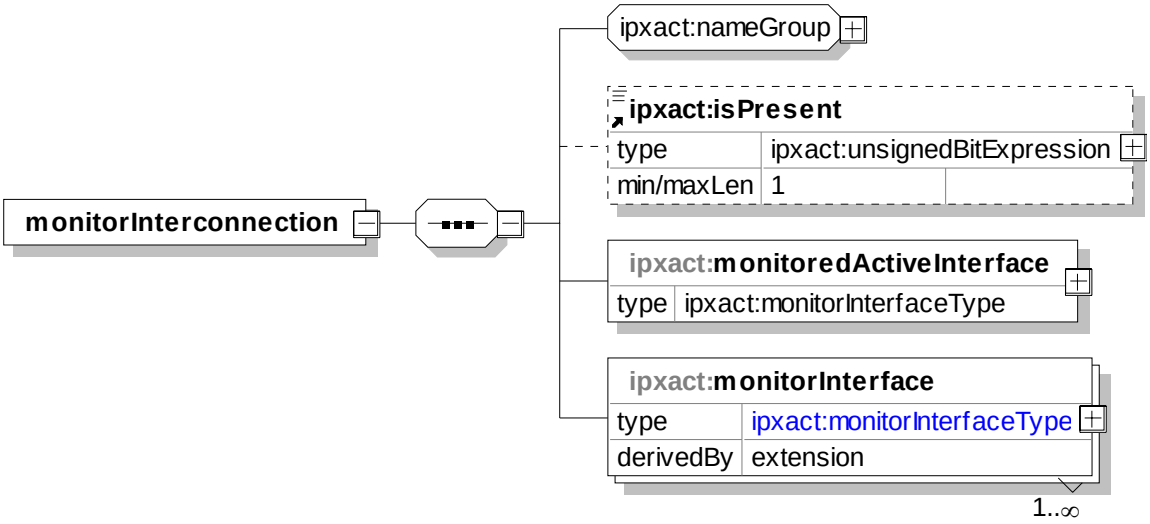
See also: SCR 15.15, SCR 15.16, and the SCRs in Table B.2 and Table B.11.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

7.3.2 monitorInterconnection

7.3.2.1 Schema

The following schema details the information contained in the **monitorInterconnection** element, which may appear as an element inside the **interconnections** element.



7.3.2.2 Description

The **monitorInterconnection** element specifies the connection between a monitored active interface on a component and a list of monitor interfaces on component instances.

a) ***nameGroup*** group is defined in C.12. The **name** elements shall be unique within the containing **interconnections** element.

b) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

c) **monitoredActiveInterface** (mandatory; type: ***moniterInterfaceType*** (see 7.4)) specifies the component bus interface to monitor.

d) **monitorInterface** (mandatory; type: ***moniterInterfaceType*** (see 7.4)) specifies the component bus interface that will do the monitoring. There may be one or more **monitorInterface** elements specified.

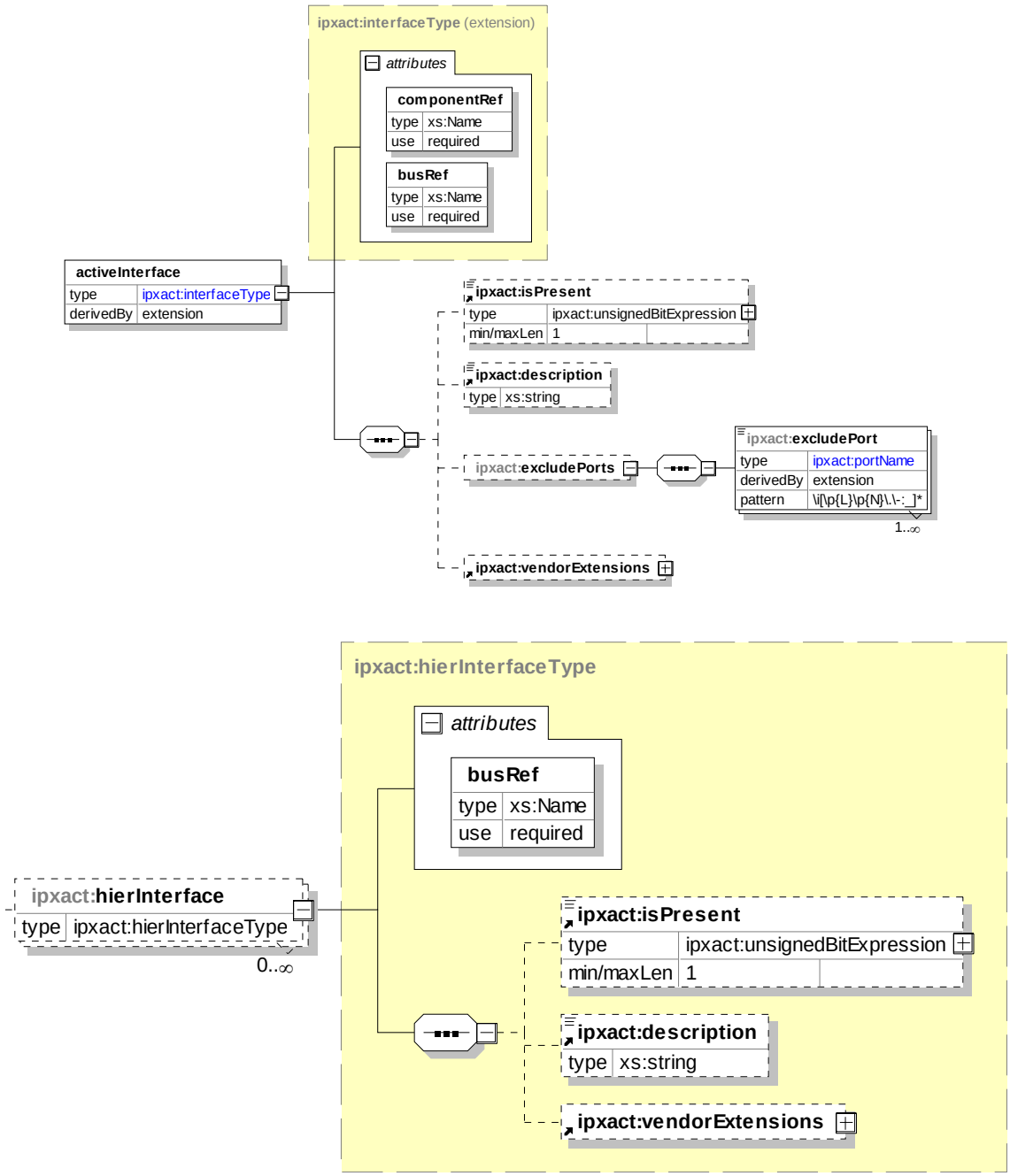
See also: SCR 6.9, SCR 6.10, SCR 15.19, SCR 15.20, SCR 15.21, SCR 15.22, and the SCRs in Table B.2 and Table B.4.

7.4 Active, hierarchical, monitored, and monitor interfaces

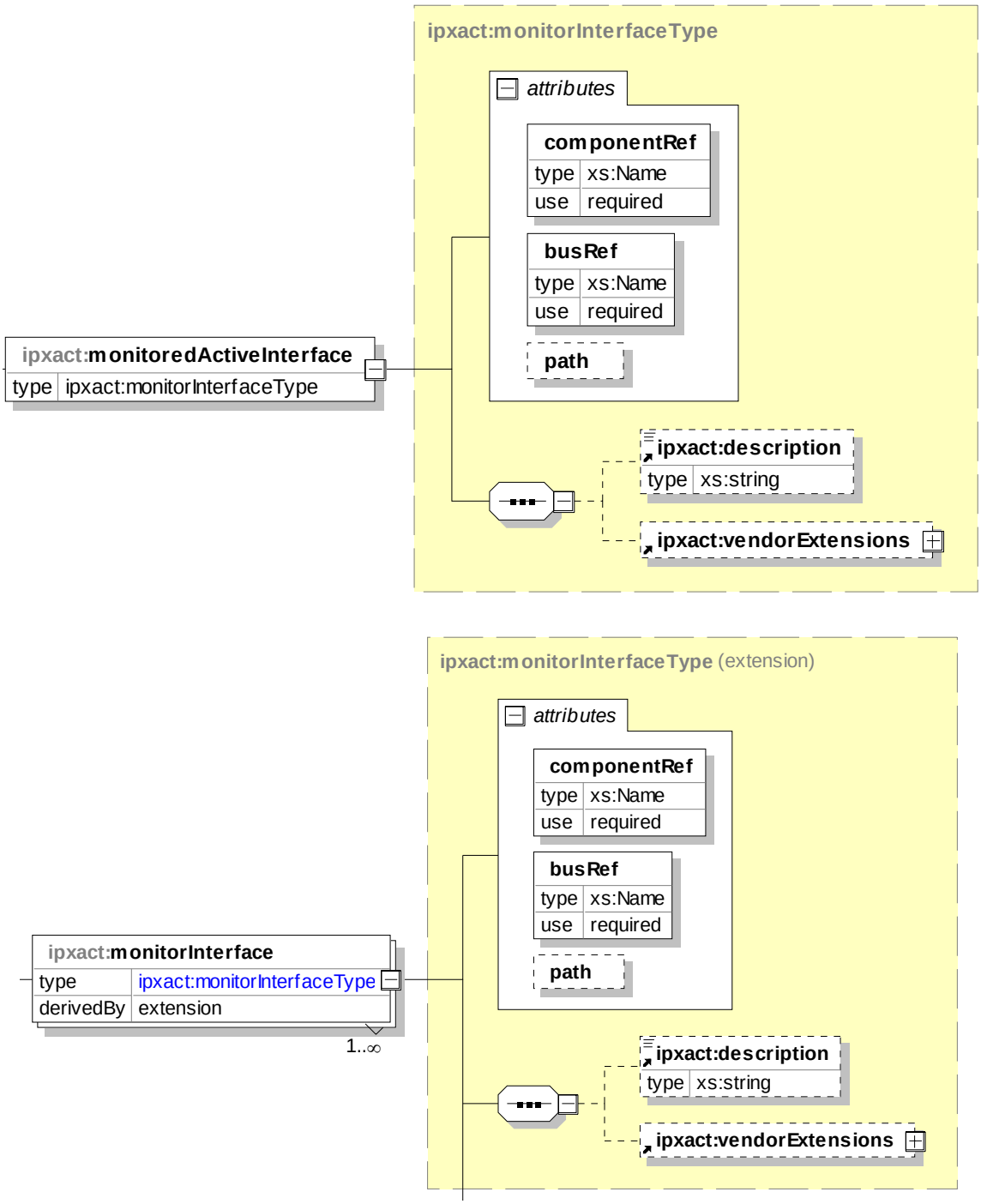
7.4.1 Schema

The following schemas define the information contained in the **activeInterface** element, the **hierInterface** element, the **monitoredActiveInterface** element, and the **monitorInterface** element, which may appear as an element inside the **interconnection** or **monitorInterconnection** element within the **interconnections** element.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



7.4.2 Description

The **activeInterface**, **hierInterface**, **monitoredActiveInterface**, or **monitorInterface** elements specify the bus interface of a design component instance that is part of an interconnection or a monitor interconnection. They all have the following attributes.

a) **busRef** (mandatory; type: ***Name***) references one of the component bus interfaces. This specific bus interface needs to exist on the specified component instance. See 6.5.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

b) **description** (optional; type: ***string***) allows a textual description of the instance.

c) **vendorExtensions** (optional) adds any extra vendor-specific data related to the design. See C.24.

The **activeInterface**, **monitoredActiveInterface**, and **monitorInterface** elements have the following attributes.

d) **componentRef** (mandatory; type: ***Name***) references the instance name of a component present in the design if the path attribute is not present. This component instance name needs to exist in the specified design. See 6.1.

The **activeInterface** and **hierInterface** elements have the following element.

e) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

The **activeInterface** element also has the following element.

f) **excludePort** (optional) contains a list of **excludePort** elements, each of which defines a logical port that should be excluded from interface driven physical connections. In effect, ports listed here are treated as if they are not defined in the **portMap** of the referenced interface.

Names listed need to exist in a **portMap** of the referenced interface.

The **monitoredActiveInterface** and **monitorInterface** elements have the following attribute.

g) **path** (optional; type: ***instancePath***) defines the hierarchical path of instance names to the design that contains the component instance specified in the **componentRef** attribute. The path is a slash (/) separated list of instance names. If the **path** attribute is not present, the component referenced by **componentRef** needs to exist in the current design. See D.4.

See also: SCR 2.1, SCR 2.16, SCR 2.18, SCR 4.1, SCR 4.2, and SCR 15.10.

7.5 Design ad hoc connections

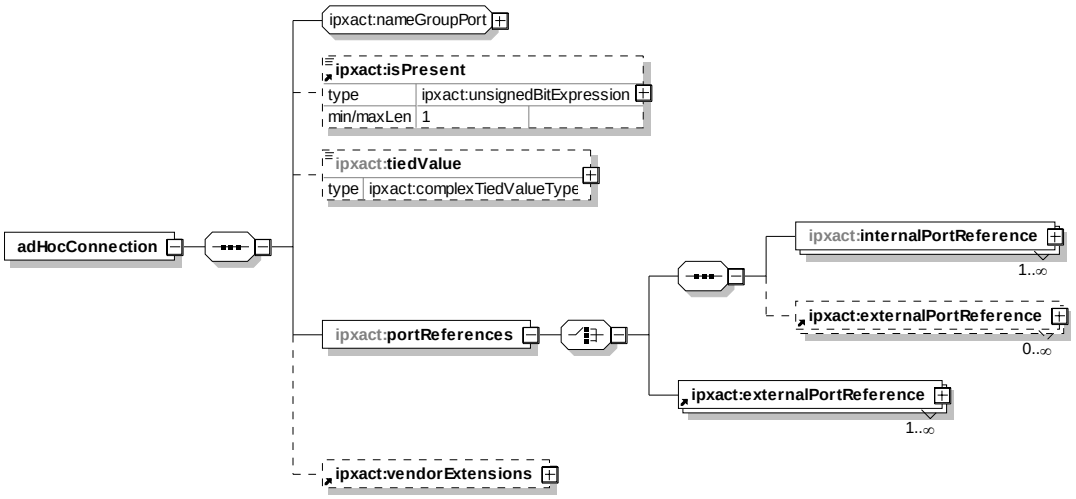
The name *ad hoc* is used for connections that are made on a port-by-port basis and not done through the higher-level bus interface. The same **ports** that make up a **busInterface** can be used in ad hoc connections.

IP-XACT supports two cases of ad hoc connections: the wire connection (between ports having a wire style) and the transactional connection (between ports having a transactional style). The direct connection between a wire-style port and a transactional-style port is not allowed; a specific adapter component needs to be inserted in between them.

7.5.1 Schema

The following schema details the information contained in the **adHocConnection** element, which may appear as an element inside the top-level **design**/**adHocConnections** element.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



7.5.2 Description

An **adHocConnection** specifies connections between component instance ports and/or between component instance ports and ports of the encompassing component (in the case of a hierarchical component). Each **adHocConnection** shall contain at least one port reference (internal or external). The **adHocConnection** element contains the following subelements.

a) ***nameGroupPort*** group is defined in C.15. The **name** elements shall be unique within the containing **adHocConnections** element.

b) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

c) **tiedValue** (optional) specifies a fixed logic value for this connection. The value of this element can be an expression which evaluates to an explicit numeric value, or the string values **default** or **open**. The value **default** implies each port should be connected to the default value defined in the port declaration. The value **open** implies each port is being intentionally left open (or unconnected). The **tiedValue** element is of type ***complexTiedValueType***.

d) **portReferences** (mandatory) specifies a list on internal and external port references for this **adHocConnection**. At least one reference needs to be defined; the following combinations are possible: one or more **internalPortReference** elements plus zero or more **externalPortReference** elements or one or more **externalPortReference** elements.

   1) **internalPortReference** references the port of a component instance. See 7.6.

   2) **externalPortReference** references a port of the encompassing component where this design is referred (for hierarchical ad hoc connections). See 7.6.

e) **vendorExtensions** (optional) adds any extra vendor-specific data related to the design. See C.24.

See also: SCR 6.26, SCR 15.23, SCR 15.24,and SCR 15.25.

7.5.3 Ad hoc wire connection

For ad hoc connections between wire-style ports, IP-XACT requires:

— The style of each port be the same style (i.e., **wire**).

— The bits of the ports are connected from left to right. See also: 7.6.

— The number of bits being connected from each referenced port shall be the same.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

See also: SCR 6.9 and SCR 6.25.

7.5.4 Ad hoc transactional connection

For ad hoc transactional connections, IP-XACT requires the style of each port be the same style (i.e., **transactional**).

See also: SCR 6.10 and SCR 6.23.

7.5.5 Interaction rules between an interface-based connection and ad hoc connections

This subclause details the rules defining the interaction between interface based connection and ad hoc connections. These rules apply to wire ports and transactional ports. Here, the term *pin* is used to designate a component port, regardless of its style.

Clause A—A pin `P` is involved in an interface based connection whenever it appears in the port map of the interface (linked to logical port `L`), and one of the following is also true:

- The logical port `L` on the opposite (connected) interface appears in the port map, meaning there is an actual pin to connect to on the opposite side of the interface.
- The definition of the logical port `L` (in its abstraction definition) contains the definition of a default value. In this case the pin will be connected to that default value.

Clause B—Connections specified via the **adHocConnection** element are always made in addition to any connections implied by interface connections.

Clause C—All connections to a pin are either made as the result of an interface connection or an ad hoc connection.

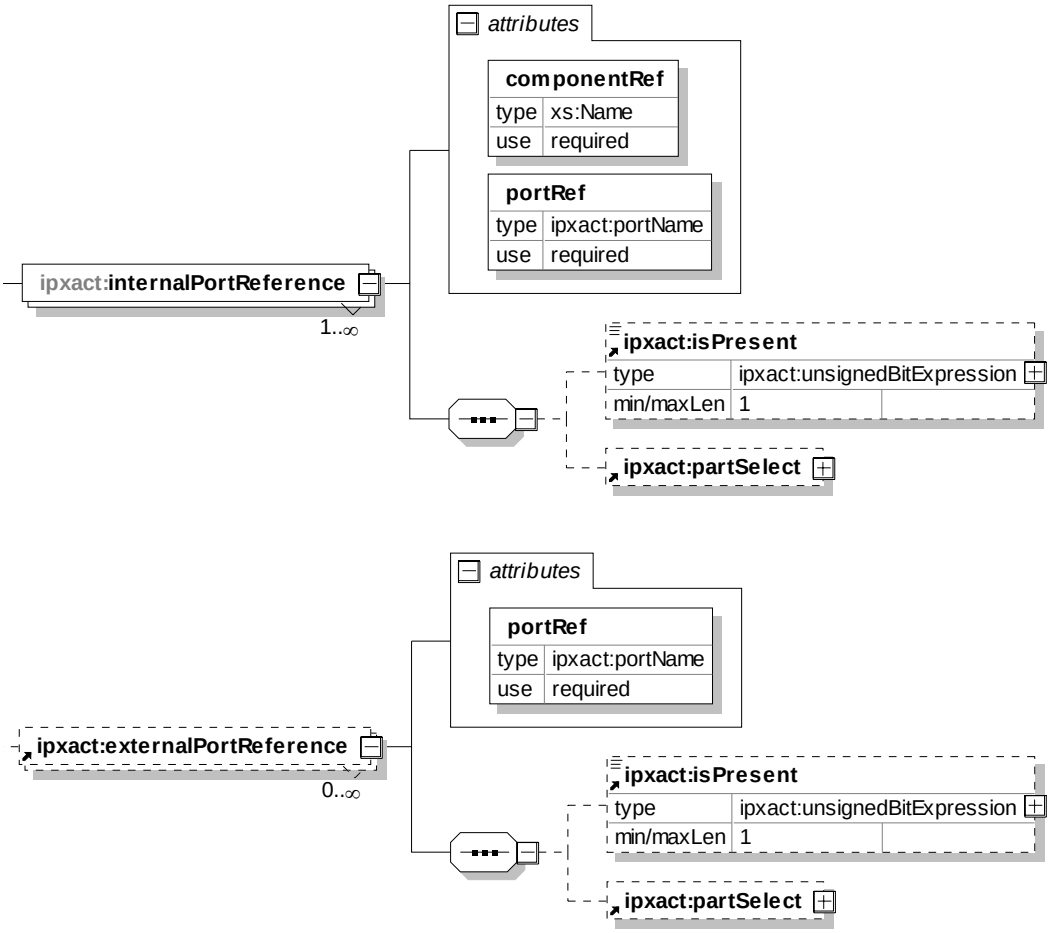
Clause A documents when a pin is considered connected via an interface. *Clause B* documents when a pin is conserved connected due to inclusion in an ad hoc connection. *Clause C* states that clauses *A* and *B* are the only ways to cause a connection to occur. Specifically, this means a pin is not considered connected from a component to its default value unless it appears in an ad hoc connection as `tiedValue == 'default'`.

7.6 Port references

7.6.1 Schema

The following schema details the information contained in the **internalPortReference** or **externalPortReference** elements, which may appear as an element within the **adHocConnections** element.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



7.6.2 Description

The **portReferences** element specifies the list of internal and external port references for this **adHocConnection**. This defines the full set of pins and ports involved in a single ad-hoc connection. **adHocConnection** has the following elements.

a) **internalPortReference** references the port of a component instance. It has the following subelements.

   1) **componentRef** (mandatory; type: ***Name***) references the component instance name for the port. See 6.1.

   2) **portRef** (mandatory; type: ***Name***) references the port name on the specific component instance. See 6.12.7.

   3) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

   4) **partSelect** (optional) defines the sub-range of the port being referenced. If no **partSelect** is specified, the entire port is assumed to be connected. See C.20.

b) **externalPortReference** references a port of the encompassing component where this design is referred (for hierarchical ad hoc connections). It has the following subelements.

   1) **portRef** (mandatory; type: ***Name***) references the port name on the encompassing component. See 6.12.7.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

2) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

3) **partSelect** (optional) defines the sub-range of the port being referenced. If no **partSelect** is specified, the entire port is assumed to be connected. See C.20.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

8. Abstractor descriptions

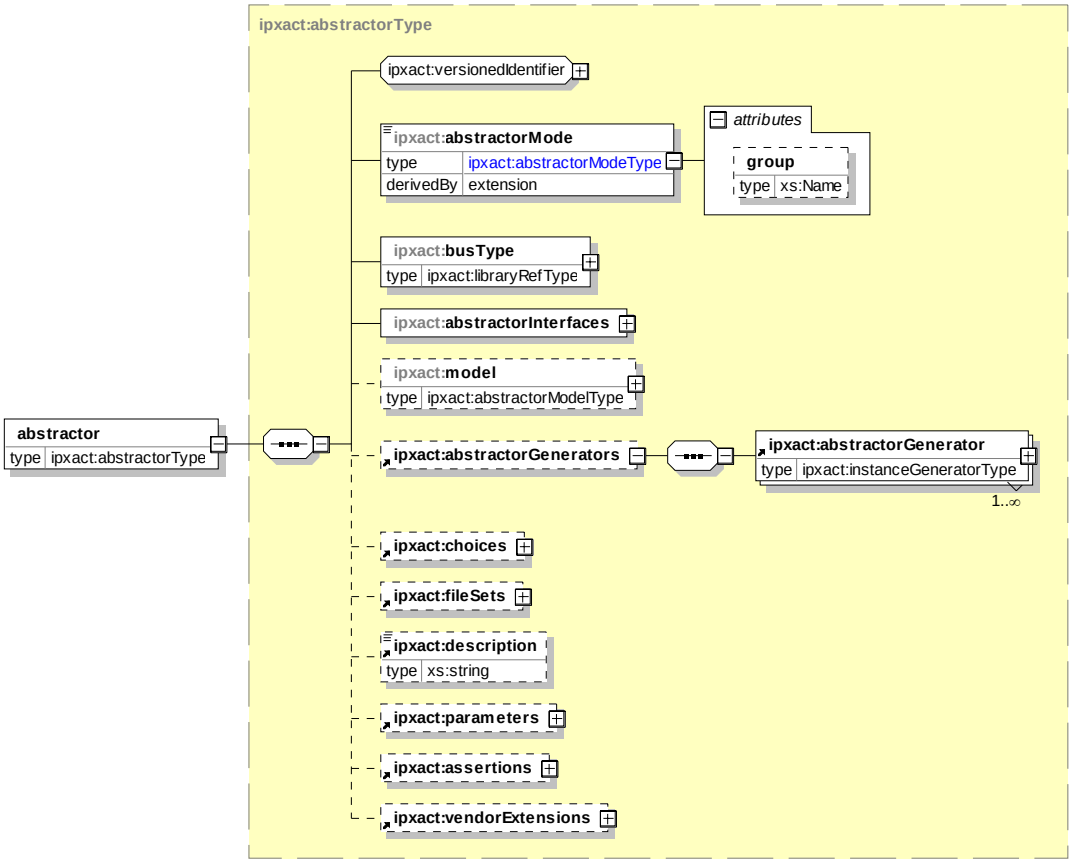
Designs that incorporate IP models using different interface modeling styles (e.g., TLM and RTL modeling styles) may contain interconnections between such component interfaces using different abstractions of the same bus type. An IP-XACT description may describe how such interconnections are to be made using a special-purpose object called an *abstractor*. An abstractor is used to connect between two different abstractions of the same bus type (e.g., an APB_RTL and an APB_TLM). An abstractor shall only contain two interfaces, which shall be of the same bus definition and different abstraction definitions.

Unlike a component, an abstractor is not referenced from a design description, but instead is referenced from a design configuration description. See Clause 10.

8.1 Abstractor

8.1.1 Schema

The following schema details the information contained in the **abstractor** element, which is one of the top-level elements in the IP-XACT specification used to describe an abstractor.



8.1.2 Description

Each element of an **abstractor** is detailed in the rest of this clause; the main sections of an **abstractor** are:

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

a) ***versionedIdentifier*** group provides a unique identifier, made up of four subelements for a top-level IP-XACT element. See C.25.

b) **abstractorMode** (mandatory) determines the mode of the two interfaces contained in **abstractorInterfaces**. The abstractor can be inserted in a connection between two instances or between an instance and an exported interface. The **abstractorMode** element can take one of the following values.

1) **master** specifies for

i) master to mirrored-master connection—the first interface connects to the master interface, the second connects to the mirrored-master interface;

ii) exported master connection—the first interface connects to the master interface, the second connects to the exported interface;

iii) exported mirrored-master connection—the first interface connects to the exported interface, the second connects to the mirrored-master interface.

2) **slave** specifies for

i) mirrored-slave to slave connection—the first interface connects to the mirrored-slave interface, the second connects to the slave interface;

ii) exported slave connection—the first interface connects to the exported interface, the second connects to the slave interface;

iii) exported mirrored-slave connection—the first interface connects to the mirrored-slave interface, the second connects to the exported interface.

3) **direct** specifies the first interface connects to the master interface, the second connects to the slave interface. This option is not allowed for an exported interface.

4) **system** specifies for

i) system to mirrored-system connection—the first interface connects to the system interface, the second connects to the mirrored-system interface;

ii) exported system connection—the first interface connects to the system interface, the second connects to the exported interface;

iii) exported mirrored-system connection—the first interface connects to the exported interface, the second connects to the mirrored-system interface.

The **group** (mandatory, when **abstractorMode** is **system**; type ***Name***) attribute defines the name of the group to which this system interface belongs. The value of this **group** shall be unique inside the **abstractor** element and already defined in the referenced abstraction definition. A connection between a **system** and **mirroredSystem** interfaces shall have matching group names.

c) **busType** (mandatory; type: ***libraryRefType*** (see C.11)) specifies the bus definition this bus interface references. A bus definition (see 5.2) describes the high-level attributes of a bus description.

d) **abstractorInterfaces** (mandatory) are interfaces having the same bus type, but differing abstraction types. See 8.2.

e) **model** (optional) specifies all the different views, ports, and model configuration parameters of the abstractor. See 8.3.

f) **abstractorGenerators** (optional) specifies a list of generator programs attached to this abstractor. See 8.7.

g) **choices** (optional) specifies multiple enumerated lists, which are referenced by other sections of this abstractor description. See 6.14.

h) **fileSets** (optional) specifies groups of files and possibly their function for reference by other sections of this abstractor description. See 6.15.

i) **description** (optional; type: ***string***) allows a textual description of the abstractor.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

j) **parameters** (optional) describes any **parameter** that can be used to configure or hold information related to this abstractor. See C.18.

k) **assertions** (optional) contains a list of expressions defining the allowed parameter values. See C.2.

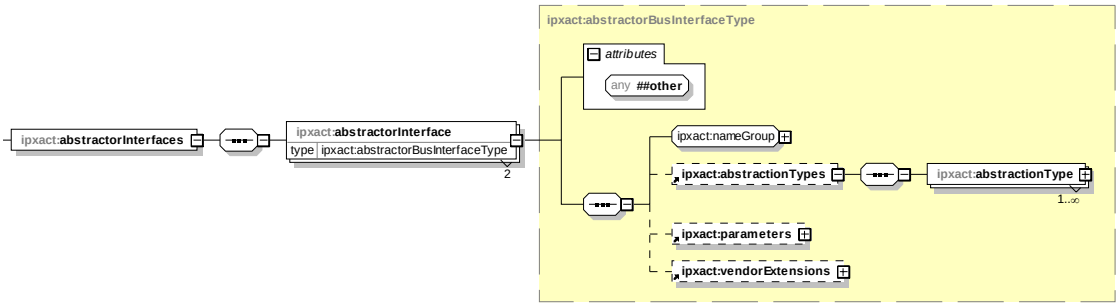
l) **vendorExtensions** (optional) contains any extra vendor-specific data related to the abstractor. See C.24.

See also: SCR 1.10, SCR 1.11, and SCR 3.15.

8.2 Abstractor interfaces

8.2.1 Schema

The following schema defines the information contained in the **abstractorInterfaces** element, which appears within an **abstractor** description.



8.2.2 Description

The **abstractorInterfaces** element contains a list of two **abstractorInterface** elements. Each **abstractorInterface** element defines properties of this specific interface in an abstractor. The **abstractorInterface** element also allows for vendor attributes to be applied. Each **abstractorInterface** contains the following elements.

a) ***nameGroup*** group is defined in C.12. The **name** elements shall be unique within the containing **abstractor** element.

b) **abstractionTypes** (optional) specifies the abstraction definition where this bus interface is referenced. An abstraction definition (the **abstractionType** element, see 6.5.6) describes the low-level attributes of a bus description (see 5.3).

c) **parameters** (optional) specifies any parameter data value(s) for this bus interface. See C.18.

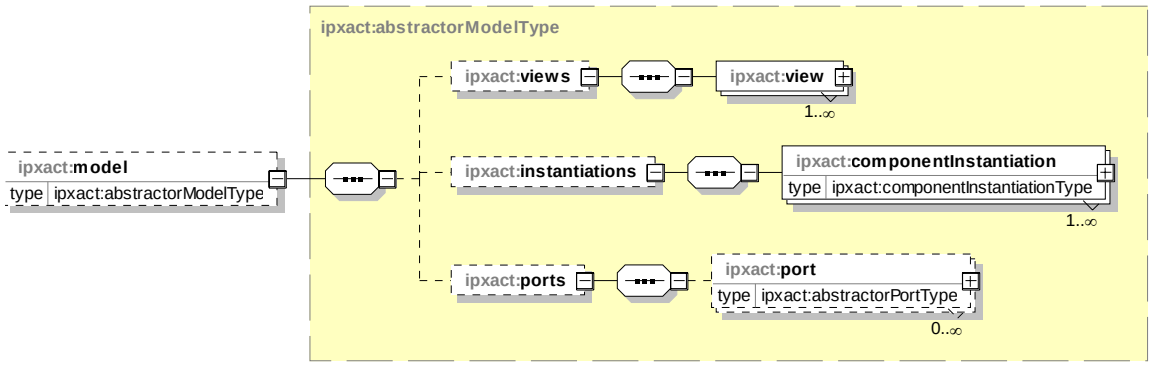
d) **vendorExtensions** (optional) holds any vendor-specific data from other namespaces, which is applicable to this bus interface. See C.24.

8.3 Abstractor models

8.3.1 Schema

The following schema defines the information contained in the abstractor **model** element, which may appear within an **abstractor** description.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



8.3.2 Description

The **model** element describes the views, ports, and model related parameters of an abstractor. A **model** element may contain the following.

a) **views** (optional) contains a list of all the views for this object. An object may have many different views. An RTL view may describe the source hardware module/entity with its pin interface; a software view may define the source device driver C file with its `.h` interface; a documentation view may define the written specification of this IP. See 8.4.

b) **instantiations** (optional) specifies the component instantiations for all views (as an ***instantiationsGroup***). See 6.12.2.

c) **ports** (optional) contains the list of ports for this object. A ports is an external connection from the object. An object may only have one set of ports that shall be valid for all views. See 8.5.

8.4 Abstractor views

8.4.1 Schema

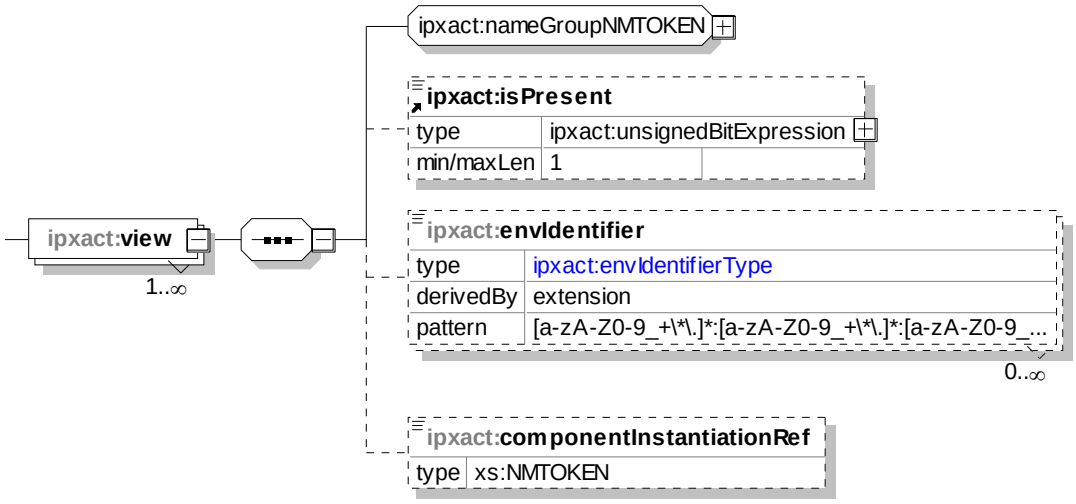
The following schema defines the information contained in the **views** element, which appears within the **model** element of an **abstractor** description.

This schema is almost identical to the **component/model/views/view** element (see 6.12.1), except:

— Abstractors have no **designInstantiationRef** elements.

— Abstractors have no **designConfigurationInstantiationRef** elements.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



8.4.2 Description

A **views** element describes an unbounded set of **view** elements. Each **view** element specifies a representation level of an abstractor. It contains the following elements.

a) ***nameGroupNMToken*** group is defined in C.16. The **name** elements shall be unique within the containing **views** element.

b) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

c) **envIdentifier** (optional) designates and qualifies information about how this model view is deployed in a particular tool environment. See 6.12.1.2.

d) **componentInstantiationRef** (optional) specifies a component's configuration instantiation for a particular instantiation. See 6.12.3.2.

See also: SCR 6.27.

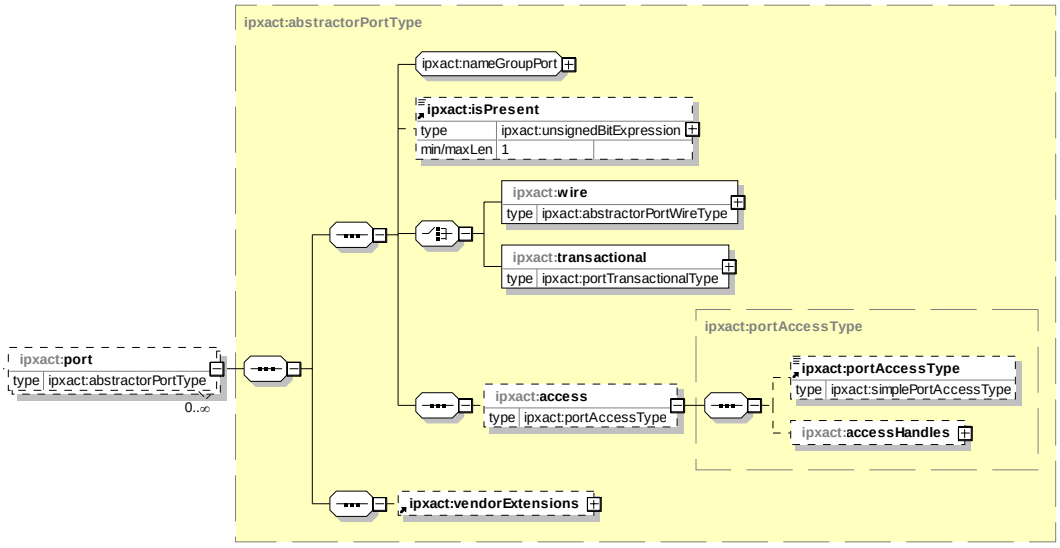
8.5 Abstractor ports

8.5.1 Schema

An abstractor's **ports** are almost identical to a component's **ports**; the abstractor **transactional** ports are exactly the same as the component **transactional** ports. The access methods are the same for an abstractor or component port. The abstractor **wire** ports defined here only differ from component **wire** ports by the absence of the **constraintSet** element, because implementation constraints are not needed for abstractors.

The following schema defines the information contained in the **ports** element, which may appear within an **abstractor**.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



8.5.2 Description

The **ports** element defines an unbounded list of **port** elements. Each **port** element describe a single external port on the abstractor.

a) ***nameGroupPort*** group is defined in C.16. The **name** elements shall be unique within the containing **ports** element.

b) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

c) Each **port** shall be described as a **wire** or **transactional** port.

1) **wire** defines ports that transport purely binary values or vectors of binary values. A wire port in an abstractor contains most of the same elements and attributes as a wire port in a component, except for the **constraintSet** element. See 8.6.

2) **transactional** defines all other style ports, typically used for TLM. A transactional port in an abstractor contains all the same elements and attributes as a transactional port in a component. See 6.12.17.

d) **access** (optional) defines the access for a port.

1) **portAccessType** (optional; default: **ref**) indicates to a netlister how to access the port. The **portAccessType** shall have one of two possible values **ref** or **ptr**. If **ref**, a netlister should access the port directly, and if **ptr**, it should access the port with a pointer.

2) **accessHandles** (optional; type: ***string***) indicates to a netlister the method to be used to access the object representing the port. This is typically a function call or array element reference in IEEE Std 1666 [B4] (SystemC).

e) **vendorExtensions** (optional) adds any extra vendor-specific data related to the port. See C.24.

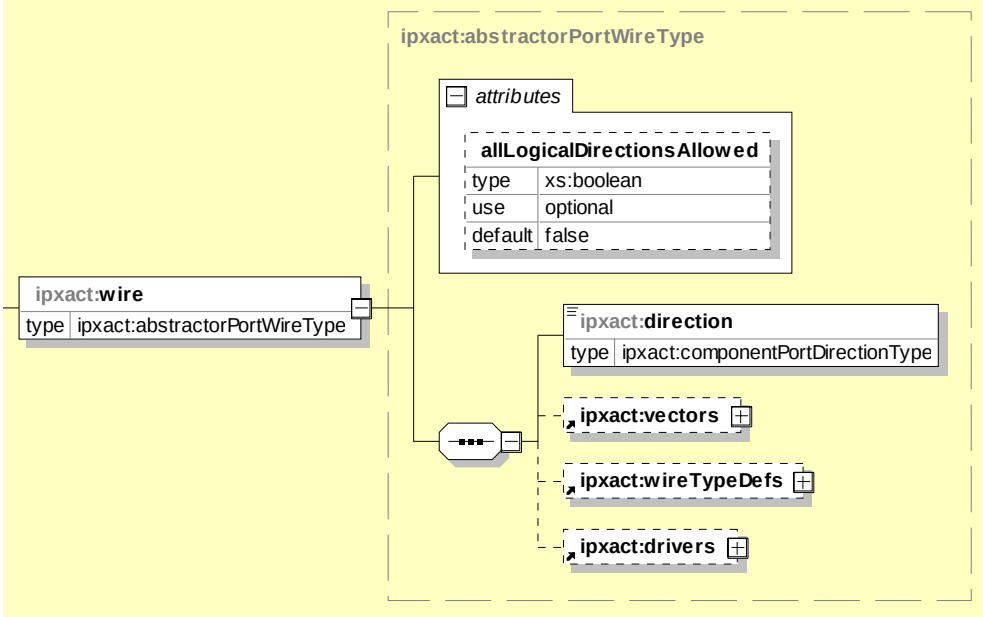
8.6 Abstractor wire ports

8.6.1 Schema

The abstractor **wire** ports defined here only differ from component **wire** ports by the absence of the **constraintSet** element, because implementation constraints are not needed for abstractors.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

The following schema element defines the information contained in the **wire** element, which appears within an abstractor **port**.



8.6.2 Description

The **wire** element describes the properties for ports that are of a wire style. A port can come in two different styles, wire or transactional. A wire port applies for all scalar types (e.g., VHDL `std_logic` and Verilog `wire`) and vectors of scalars. A wire port transports purely binary values or vectors of binary values.

— Scalar types in VHDL also include integer and enumeration values. Scalars in IP-XACT only include binary values that relate to a single wire in a hardware implementation.

— Since wire ports allow only binary values, IP-XACT does not support tri-state or multiple strength values.

The **wire** element contains the following elements.

a) **allLogicalDirectionsAllowed** (optional; type: ***boolean***; default: **false**) attribute defines whether the port may be mapped to a port in an **abstractionDefinition** with a different direction. See 5.3.

b) **direction** (mandatory) specifies the direction of this port: **in** for input ports, **out** for output ports, and **inout** for bidirectional and tri-state ports. **phantom** can also be used to define a port that only exists on the IP-XACT component, but not on the implementation referenced from the view.

c) **vectors** (optional) specifies the dimensions for a non-scalar port, see C.23.

d) **wireTypeDefs** (optional) describes the ports type as defined by the implementation, see 6.12.9.

e) **drivers** (optional) defines an unbounded list of driver elements, defining drivers that may be attached to this port if no other object is connected to this port. This allows the IP to define the default state of unconnected inputs. A wire style port may only define a **driver** element for a port if the direction of the port is **in** or **inout**. See also 6.12.10.

   If multiple drivers are specified, the **range** element shall be used to indicate which bits are driven by which driver and no overlap bit ranges may be specified.

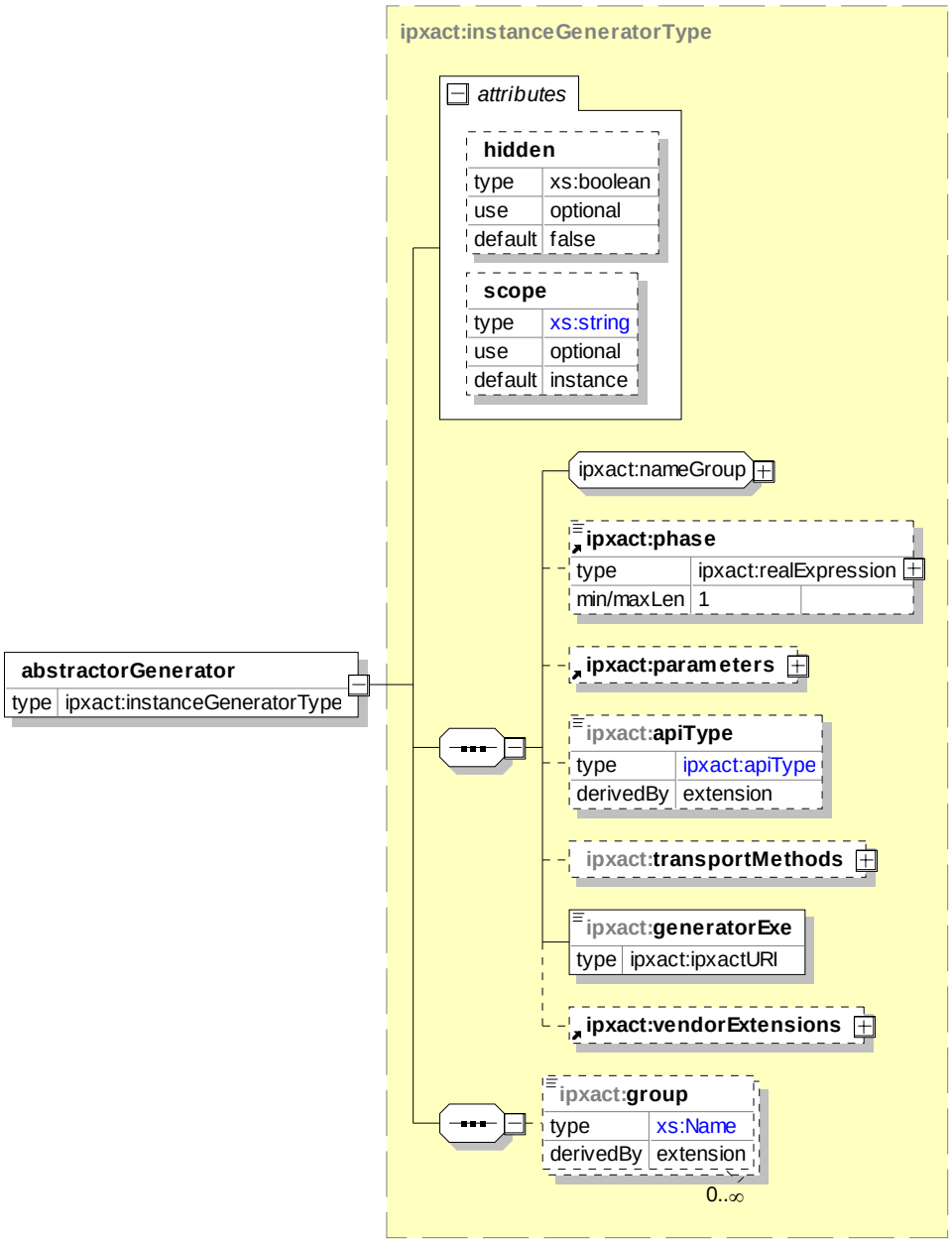
See also: SCR 6.5, SCR 6.6, SCR 6.7, and SCR 6.12.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

8.7 Abstractor generators

8.7.1 Schema

The following schema defines the information contained in the **abstractorGenerators** element, which may appear within an **abstractor** object.



Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

8.7.2 Description

The **abstractorGenerators** element contains an unbounded list of **abstractorGenerator** elements. Each **abstractorGenerator** element defines a generator that is assigned and may be run on this abstractor. The **abstractorGenerator** has exactly the same schema definition as a **componentGenerator**. See 6.13.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

9. Generator chain descriptions

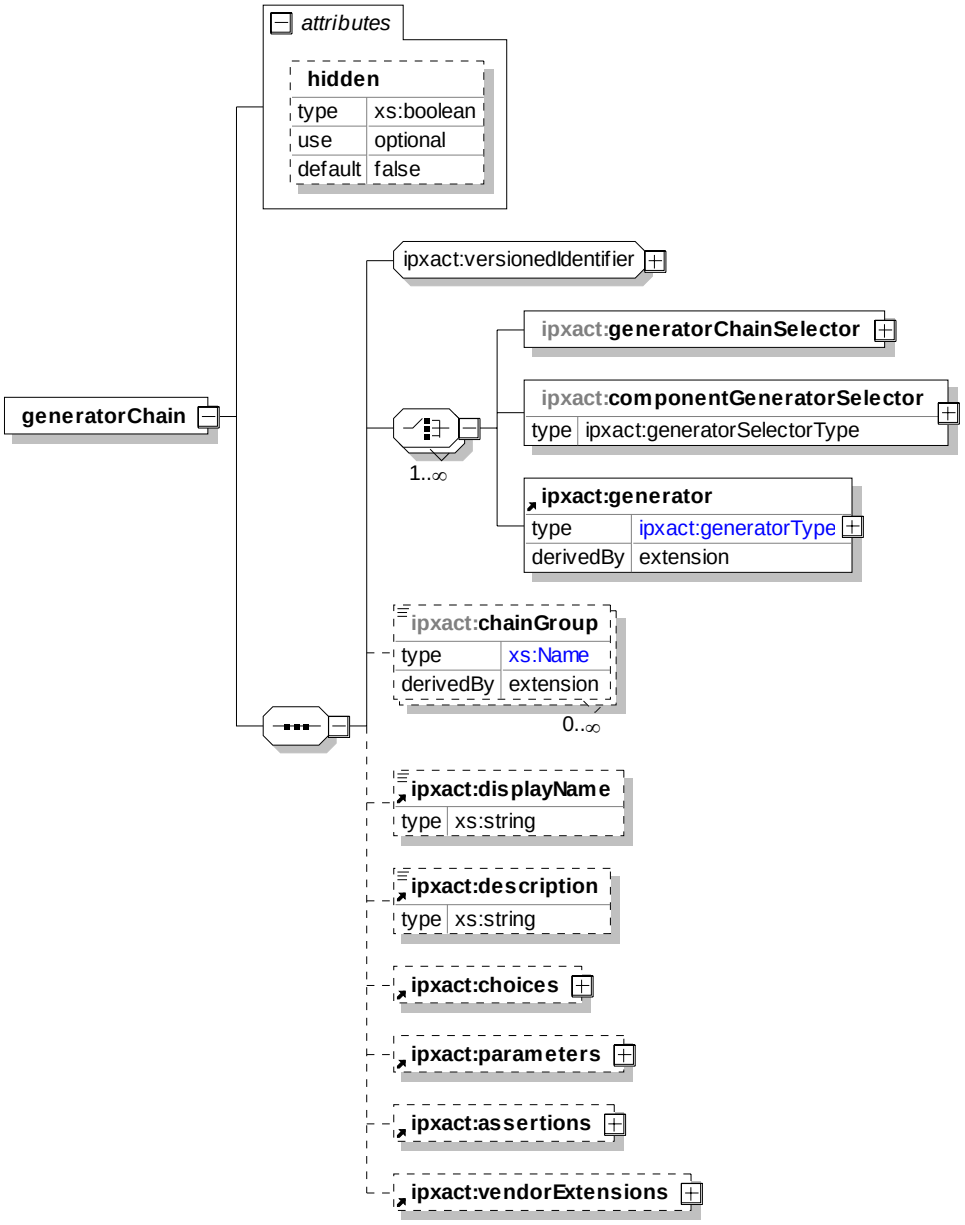
9.1 generatorChain

In IP-XACT, a design flow can be represented as a generator chain. A generator chain is an ordered sequence of named tasks. Each named task can be represented as a single generator or as another generator chain. This way, design flow hierarchies can be constructed and executed from within a given DE. The DE itself is responsible for understanding the semantics of the specified chain described in the generator chain description.

9.1.1 Schema

The following schema details the information contained in the **generatorChain** element, which is one of the top-level elements in the IP-XACT specification.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



9.1.2 Description

The **generatorChain** element describes a single generator chain. The **generatorChain** element has a **hidden** attribute (optional; type: ***boolean***; default: **false**) attribute that, when **true**, indicates this generator chain is not presented to the user of a DE. This may be the case if the chain is part of another chain and has no useful meaning when invoked as stand-alone. The **generatorChain** element contains the following elements.

a) ***versionedIdentifier*** group provides a unique identifier; it consists of four subelements for a top-level IP-XACT element. See C.25.

b) A **generatorChain** also contains one or more of the following subelements.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

1) **generatorChainSelector** is a selection criteria for selecting one or more **generatorChain**s or a reference to another **generatorChain** (see 9.2).

2) **componentGeneratorSelector** is a selection criteria for selecting one or more component generators (see 9.3).

3) **generator** defines the generator (see 9.4).

c) **chainGroup** (optional; type: ***Name***) is an unbounded list of names to which this chain belongs. The group names are referenced in the **generatorChainSelector** element and can be used to organize the inclusion of generators.

d) **displayName** (optional; type: ***string***) allows a short descriptive text to be associated with the generator chain.

e) **description** (optional; type: ***string***) allows a textual description of the generator chain.

f) **choices** (optional) specifies multiple enumerated lists, which are referenced by other sections of this generator chain description. See 6.14.

g) **parameters** (optional) describes any **parameter** that can be used to configure or hold information related to this generator chain. See C.18.

h) **assertions** (optional) contains a list of expressions defining the allowed parameter values. See C.2.

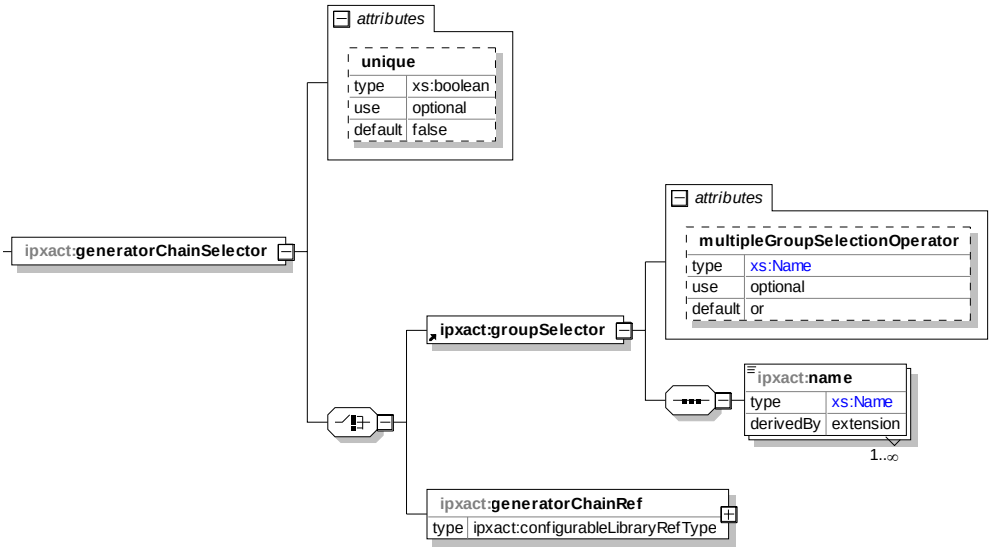
i) **vendorExtensions** (optional) contains any extra vendor-specific data related to the **generatorChain**. See C.24.

See also: SCR 1.10.

9.2 generatorChainSelector

9.2.1 Schema

The following schema defines the information contained in the **generatorChainSelector** element, which may appear within a **generatorChain**.



Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

9.2.2 Description

The **generatorChainSelector** element defines which generator(s) to invoke based on a selection criteria. The **generatorChainSelector** element contains a **unique** (optional; type: ***boolean***; default: **false**) attribute that, when **true**, indicates the generatorChainSelector shall resolve to a single generator. If more than one generator is selected, the DE shall resolve the selection to a single generator.

The **generatorChainSelector** element can specify the selection criteria in one of two ways: as a selection based on the **chainGroup** names via the **groupSelector** element or as a direct VLNV reference via the **generatorChainRef** element. The **generatorChainSelector** element shall contain one of the **groupSelector** or **generatorChainRef** elements.

a) **groupSelector** is a container for an unbounded list of chain group **name** elements.

   1) When more than one **name** element is specified, the **multipleGroupSelectorOperator** (optional; type: ***Name***; default: **or**) attribute can specify if the selection applies when *one* of the generator group names matches (**multipleGroupSelectorOperator** equals **or**) or *all* the generator group names match (**multipleGroupSelectorOperator** equals **and**).

   2) **name** (mandatory; type ***Name***) is an unbounded list of selection names. The names are compared to the **generatorChain/chainGroup** elements within all generator chains visible to the DE.

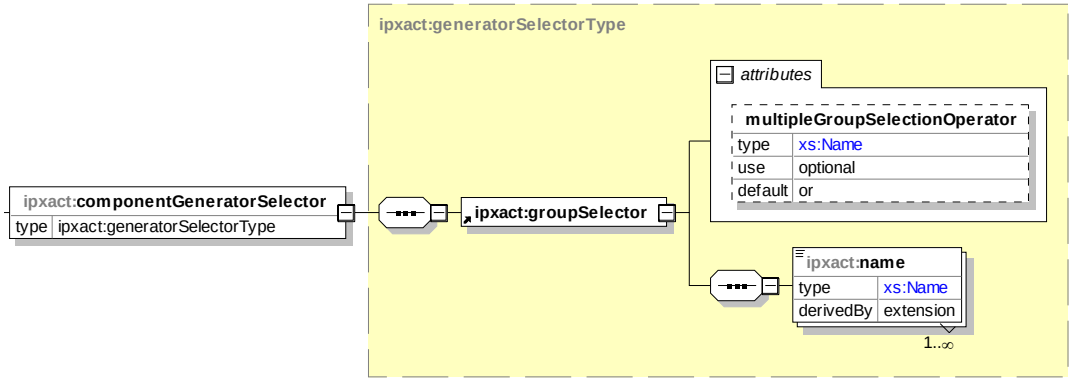
b) **generatorChainRef** (type: ***configurableLibraryRefType*** (see C.6)) specifies a reference to another generator chain description for inclusion in this generator chain.

See also: SCR 1.8.

9.3 generatorChain component selector

9.3.1 Schema

The following schema defines the information contained in the **componentGeneratorSelector** element, which may appear within a **generatorChain**.



9.3.2 Description

Similar to the **generatorChainSelector**, **componentGeneratorSelector** selects a component generator or a list of component generators based on the assigned group name. The **componentGeneratorSelector** contains the **groupSelector** element.

groupSelector (mandatory) is a container for an unbounded list of chain group **name** elements.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

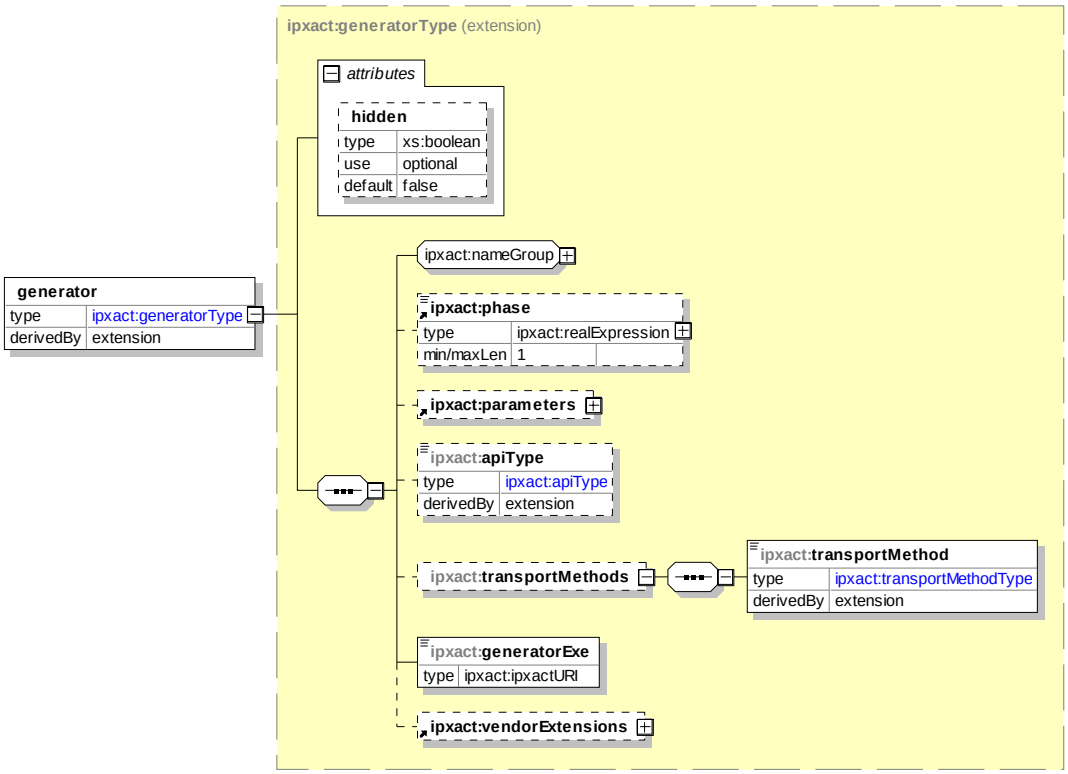
1) When more than one **name** element is specified, the **multipleGroupSelectorOperator** (optional; type: ***Name***; default: **or**) attribute can specify if the selection applies when *one* of the generator group names matches (**multipleGroupSelectorOperator** equals **or**) or *all* the generator group names match (**multipleGroupSelectorOperator** equals **and**).

2) **name** (mandatory; type: ***Name***) is an unbounded list of selection names. The names are compared to the **componentGenerator**/**group** elements within all components in the current design.

9.4 generatorChain generator

9.4.1 Schema

The following schema defines the information contained in the **generator** element, which may appear within a **generatorChain**.



9.4.2 Description

The **generator** element defines a specific generator executable. The **generator** element contains has a **hidden** attribute (optional; type: ***boolean***; default: **false**) attribute that, when **true**, indicates this generator shall not be run as a stand-alone generator and is required to be run as part of a chain. This generator is not presented to the user. If **false**, this generator may be run as a stand-alone generator or in a generator chain.

generator contains the following elements.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

a) ***nameGroup*** group is defined in C.12.

b) **phase** (optional; type: ***realExpression*** (see C.3.1)) determines the sequence in which a generators are run. Generators are run in order starting with zero (`0`). If two generators have the same phase numbers, the order shall be interpreted as not important and the generators can be run in any order. If no phase number is given the generator is considered in the "last" phase and these generators are run in any order after the last generator with a phase number. The **phase** element shall be a positive number.

c) **parameters** (optional) specifies any **generator** type parameters. See C.18.

d) **apiType** (optional, type: **apiType**, default: **TGI_2014_BASE**) indicates the type of API used by the generator. Allowed values include:

1) **TGI_2014_BASE** indicates a generator which communicates using the base TGI API defined for this Standard.

2) **TGI_2014_EXTENDED** indicates a generator which communicates using the base and extended TGI APIs defined for this Standard.

3) **TGI_2009** indicates a generator which communicates using the TGI API defined for IEEE Std 1685-2009. Such APIs are not part of this Standard and do not need to be supported by IP-XACT enabled DEs.

4) **none** indicates the generator does not communicate with the DE.

e) **transportMethods** (optional) defines alternate SOAP transport protocol that this generator can support. The default SOAP transport protocol is HTTP if this element is not present.

transportMethod specifies the alternate transport protocol. This element is an enumerated list of only one element **file**. **file** indicates the SOAP transport protocol is transported to the DE via a file or file handle.

f) **generatorExe** (mandatory; type ***ipxactURI***) contains an absolute or relative (to the location of the containing description) path to the generator executable. The path may also contain environment variables from the host system, which are used to abstract the location of the generator.

g) **vendorExtensions** (optional) adds any extra vendor-specific data related to the **generator**. See C.24.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

10. Design configuration descriptions

10.1 Design configuration

An IP-XACT *design configuration* is a placeholder for additional configuration information of a design or generator chain description. Design configuration information is used to support configuration of hierarchical designs, for transferring designs between design environments and automating generator chain execution for a design, by storing information that would otherwise have to be re-entered by the designer.

The *design configuration description* contains the following configuration information:

— Configuration values for parameters defined in generators within generator chains; this information is not referenced via the design description;

— Active view or current view selection for instances in the design description along with configuration values for view parameters within the component instances;

— Configuration information for interconnections between the same bus types with differing abstraction types (i.e., abstractor reference, parameter configuration, and view selection). See also Clause 8.

A design configuration applies to a single design, but a design may have multiple design configuration descriptions.

10.2 designConfiguration

10.2.1 Schema

The following schema details the information contained in the **designConfiguration** element, which is one of the top-level elements of the schema.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

10.2.2 Description

The **designConfiguration** element details the configuration for a design or generator chain description. The **designConfiguration** element contains the following mandatory and optional elements.

a) The ***versionedIdentifier*** group provides a unique identifier, made up of four subelements for a top-level IP-XACT element. See C.25.

b) **designRef** (optional; type: ***libraryRefType*** (see C.11)) specifies the design description for this design configuration.

NOTE—It is recommended instead to pursue this functionality by using the **designInstantiation** element in a **component**.

c) **generatorChainConfiguration** (optional; type: ***configurableLibraryRefType*** (see C.6)) documents a generator chain VLNV and an unbounded list of configuration values for parameters within the referenced generator chain.

d) **interconnectionConfiguration** (optional) is an unbounded list of information associated with interface interconnections. Any abstractors required for the connection of two interfaces are specified here. See 10.3.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

e) **viewConfiguration** (optional) defines the selected view and view configuration for an instance of the design. See 10.5.

f) **description** (optional) allows a textual description of the design configuration. The **description** element is of type ***string***.

g) **parameters** (optional) describes any **parameter** that can be used to configure or hold information related to this design configuration. See C.18.

h) **assertions** (optional) contains a list of expressions defining the allowed parameter values. See C.2.

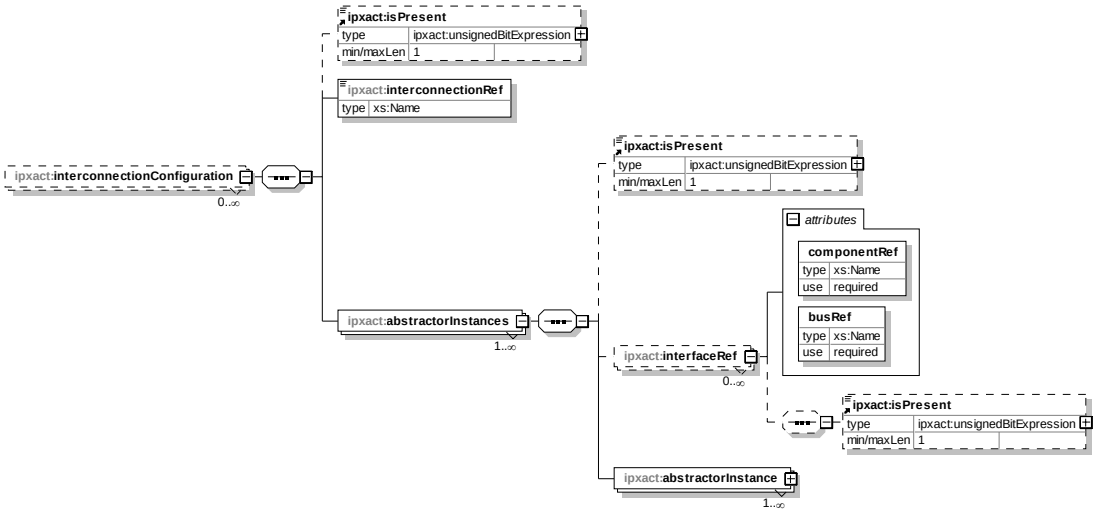
i) **vendorExtensions** (optional) adds any extra vendor-specific data related to the design configuration. See C.24.

See also: SCR 1.5, SCR 1.7, SCR 1.10, SCR 1.16, SCR 1.17, SCR 5.10, SCR 13.1, SCR 13.2, and SCR 13.3.

10.3 interconnectionConfiguration

10.3.1 Schema

The following schema defines information contained in **interconnectionConfiguration** element, which may appear as an element inside the **designConfiguration** element.



10.3.2 Description

The **interconnectionConfiguration** element contains information about the **abstractor**s used to resolve connections between interfaces having the same **busDefinition** types, but different **abstractionDefinition** types. The **interconnectonConfiguration** element contains the following elements.

a) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

b) **interconnectionRef** (mandatory; type: ***Name***) contains a reference to a design **interconnection**/**name** or a design **monitorInterconnection**/**name**. All **interconnectionRef** elements shall be unique within the containing design configuration description.

c) **abstractorInstances** (mandatory) contains an unbounded list of **abstractor** elements. Multiple abstractors elements are allowed to enable insertion of different abstractor chains within a broadcast

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

interface. If there are multiple abstractors elements, each is required to reference a unique interface via the **componentRef** and **interfaceRef** attributes in the **interfaceRef** element. If an **abstractors** element contains no **interfaceRef** element, it is presumed the contained chain of abstractors applies to all interfaces not explicitly referenced in other abstractors elements. The **abstractors** element contains the following mandatory and optional elements.

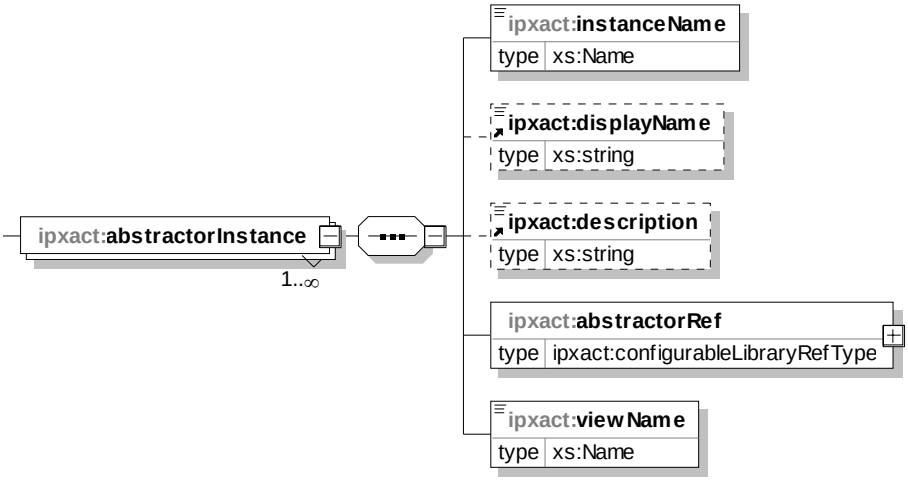
1) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.
2) **interfaceRef** (optional) defines the broadcast endpoint for this abstractor chain. It contains the following subelements.
   i) **componentRef** (mandatory; type: ***Name***) refers to a component instance name.
   ii) **busRef** (mandatory; type: ***Name***) refers to a component bus interface name.
   iii) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.
3) **abstractorInstance** (mandatory) is an ordered list for chaining the abstractors together to bridge from one abstraction to another. See 10.4.

See also: SCR 1.13, SCR 3.7, SCR 3.11, SCR 3.12, SCR 3.13, SCR 3.14, SCR 3.18, SCR 5.11, SCR 13.4, SCR 15.17, and SCR 15.18.

10.4 abstractorInstance

10.4.1 Schema

The following schema defines information contained in **abstractorInstance** element, which may appear as an element inside the **interconnectonConfiguration** element.



10.4.2 Description

The **abstractorInstance** element is an ordered list for chaining the abstractors together to bridge from one abstraction to another. This element has the following subelements.

a) **instanceName** (mandatory; type: ***Name***) assigns a unique name for this instance of the abstractor in this design. The value of this element shall be unique inside the **designConfiguration** and the referenced **design** element.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

b) **displayName** (optional; type: ***string***) allows a short descriptive text to be associated with the instance.

c) **description** (optional; type: ***string***) allows a textual description of the instance.

d) **abstractorRef** (mandatory; type: ***configurableLibraryRefType*** (see C.6)) is a reference to an abstractor description for this abstractor instance.

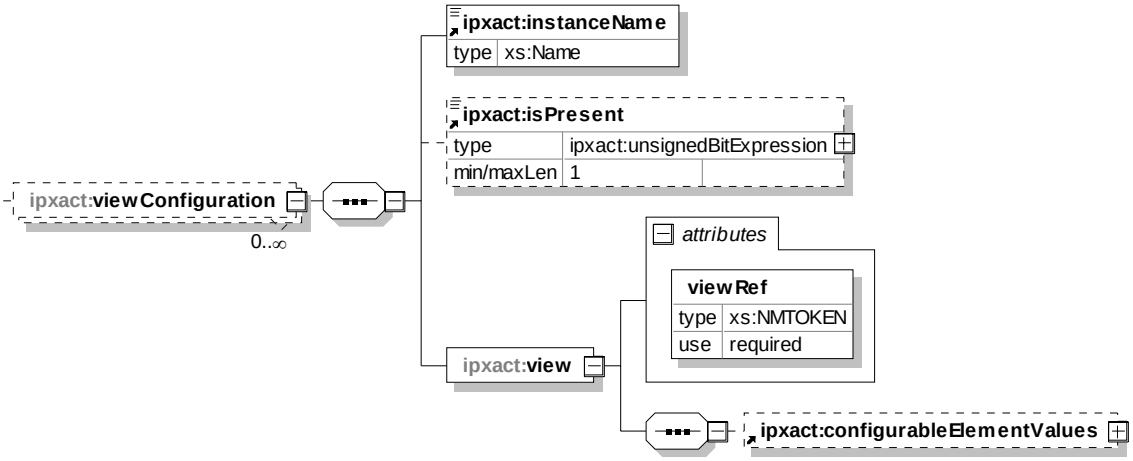
e) **viewName** (mandatory; type: ***Name***) defines the selected view for the abstractor instance.

See also: SCR 1.13, SCR 3.7, SCR 3.11, SCR 3.12, SCR 3.13, SCR 3.14, SCR 5.11, and SCR 13.4.

10.5 viewConfiguration

10.5.1 Schema

The following schema defines information contained in **viewConfiguration** element, which may appear as an element inside the **designConfiguration** element.



10.5.2 Description

The **viewConfiguration** element defines the active view and view configuration values for an instance of the design. The **viewConfiguration** element contains the following elements.

a) **instanceName** (mandatory; type: ***Name***) specifies the component instance name for which the view is being selected. This instance name shall be unique within the containing design configuration description.

b) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

c) **view** (mandatory) defines the current valid view for the selected component instance.

   1) **viewRef** (mandatory; type: ***NMTOKEN***) specifies the component view. See 6.12.1.2.

   2) **configurableElementValues** (optional) specifies the configuration values associated with parameters within the **componentInstantiation** and **designConfigurationInstantiation** referenced by the component view specified by the **viewRef**. See C.5.

See also: SCR 5.9, SCR 15.11, and SCR 15.12.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

11. Catalog descriptions

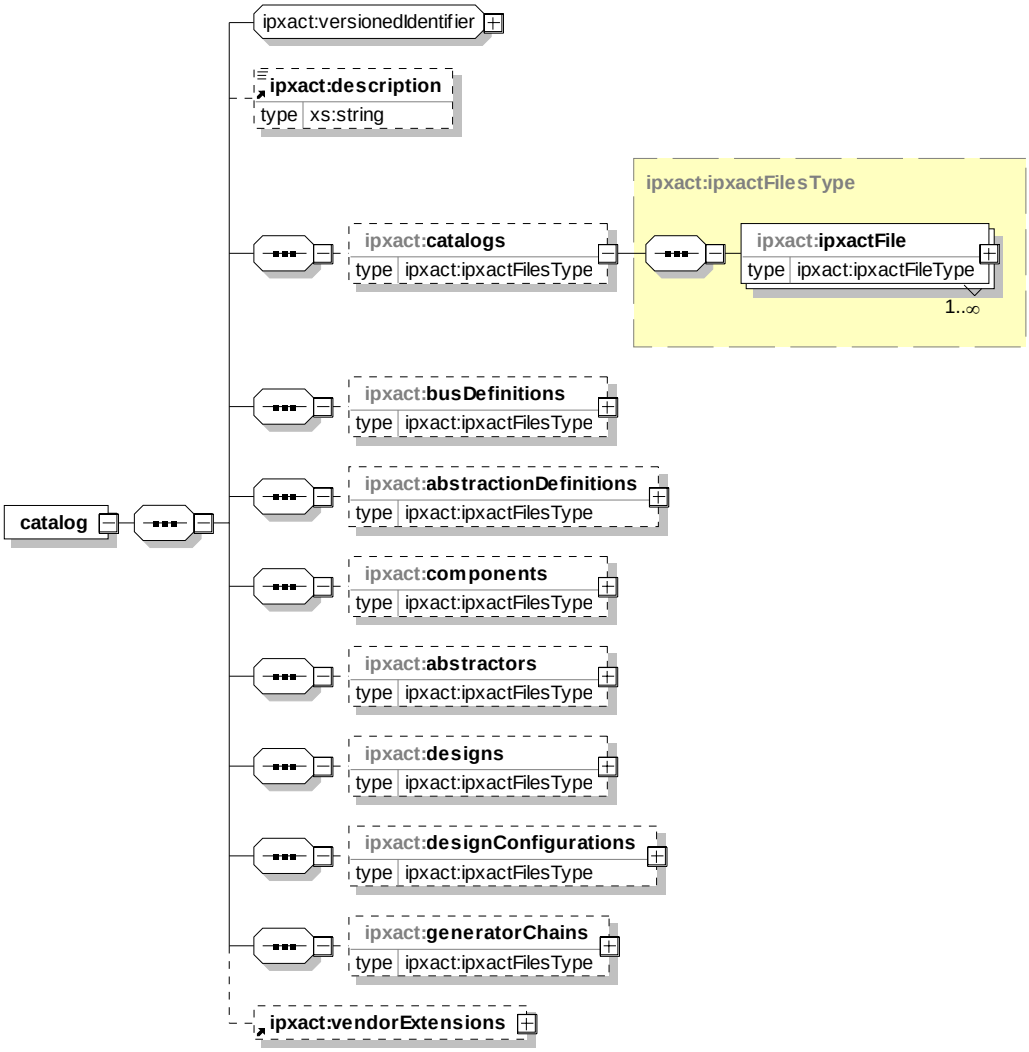
11.1 catalog

Systems described with IP-XACT can consist of a single XML file, but more often reference a large number of files (components, bus/abstraction definitions, etc.). The top-level **catalog** element provides a mechanism for documenting the locations of these files and the top-level details (VLNV and document type) giving design environments a standard means of locating IP-XACT documents.

A *catalog* can document the location, VLNV, and file type of any number of IP-XACT documents, including other catalogs.

11.1.1 Schema

The following schema details the information contained in the **catalog** element, which contains the optional list elements **catalogs**, **busDefinitions**, **abstractionDefinitions**, **components**, **abstractors**, **designs**, **designConfigurations**, and **generatorChains**.



Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

11.1.2 Description

The top-level **catalog** element references top-level IP-XACT elements and specifies where each can be found. The **catalog** contains the following elements and attributes.

a) The ***versionedIdentifier*** group provides a unique identifier; it consists of four subelements for a top-level IP-XACT element. See C.25.

b) **description** (optional; type: ***string***) allows a textual description of the interface.

c) **catalogs** (optional) specifies a list of **ipxactFile** elements that reference other IP-XACT **catalog** documents. See 11.2.

d) **busDefinitions** (optional) specifies a list of **ipxactFile** elements that reference other IP-XACT **busDefinitions** documents. See 11.2.

e) **abstractionDefinitions** (optional) specifies a list of **ipxactFile** elements that reference other IP-XACT **abstractionDefinitions** documents. See 11.2.

f) **components** (optional) specifies a list of **ipxactFile** elements that reference other IP-XACT **components** documents. See 11.2.

g) **abstractors** (optional) specifies a list of **ipxactFile** elements that reference other IP-XACT **abstractors** documents. See 11.2.

h) **designs** (optional) specifies a list of **ipxactFile** elements that reference other IP-XACT **designs** documents. See 11.2.

i) **designConfigurations** (optional) specifies a list of **ipxactFile** elements that reference other IP-XACT **designConfigurations** documents. See 11.2.

j) **generatorChains** (optional) specifies a list of **ipxactFile** elements that reference other IP-XACT **generatorChains** documents. See 11.2.

k) **vendorExtensions** (optional) contains any extra vendor-specific data related to the interface. See C.24.

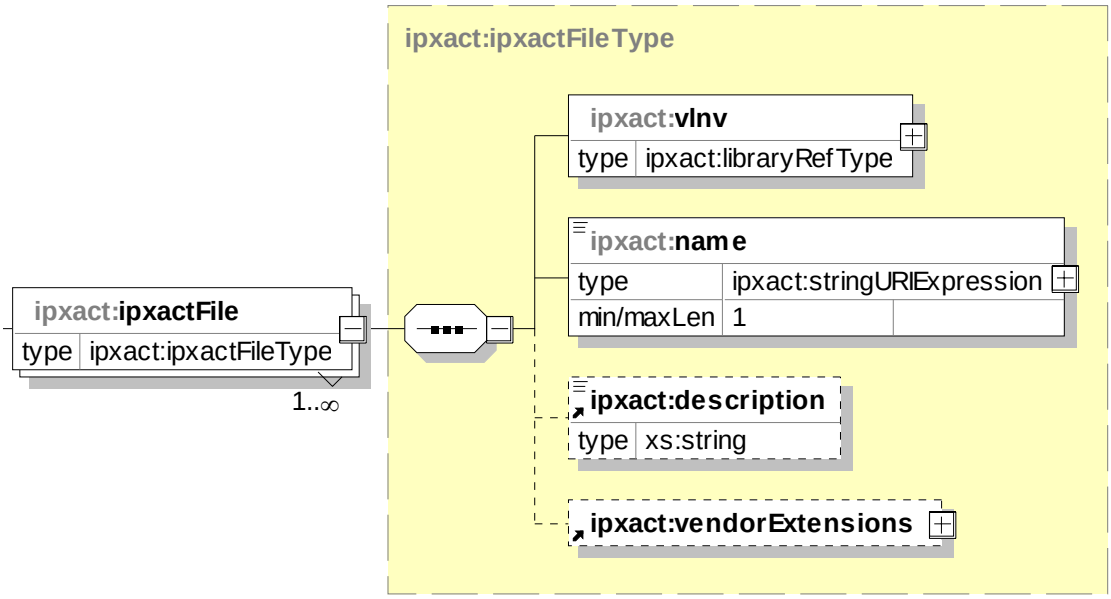
See also: SCR 1.3, SCR 1.10, SCR 1.12, and SCR 6.16.

11.2 ipxactFile

11.2.1 Schema

The following schema details the information contained in the **ipxactFile** element, which documents the existence of a top-level IP-XACT file with the indicated location and VLNV. The type of the IP-XACT file is expected to match that of the container element of the **ipxactFile**.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



11.2.2 Description

The **ipxactFile** contains the following elements and attributes.

a) **VLNV** (mandatory; type: ***libraryRefType*** (see C.11)) specifies the VLNV of the IP-XACT file referenced by the **name** element.

b) **name** (mandatory; type: ***stringURIExpression*** (see C.3.4)) identifies the containing element.

c) **description** (optional; type: ***string***) allows a textual description of the containing element.

d) **vendorExtensions** (optional) adds any extra vendor-specific data related to the design configuration. See C.24.

See also: SCR 1.15.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

12. Addressing

This clause describes how addresses are transformed between a slave's memory map and a master's address space. Clause 13 also describes how to determine which bits of the memory map are visible in the master's address space.

Indirect interfaces and the associated indirectly accessible memory map do not directly add to the address map. Rather, each indirect interface requires a separate address map calculation. Address map calculation for indirect interfaces is the same as a slave bus interface. The indirect address field and indirect data field are analogous to the logical address and data ports of a slave bus interface.

NOTE—In both Clause 12 and Clause 13, *equation variables* are denoted with letters offset in parenthesis, e.g., `(b)`, `(d)`, etc; whereas, the equations themselves are numbered (and offset in parenthesis), e.g., `(1)`, `(31)`, etc. Any subsequent references are shown as superscripted (and enclosed in parenthesis), e.g., `address_offset(a)` or `item_width(4)`.

12.1 Calculating the bit address of a bit in a memory map

A *memory map* consists of a set of address blocks, subspace maps, and banks containing further address blocks, subspace maps, and banks (to any number of levels). To calculate the address of a bit within an address block or subspace map relative to the containing memory map, its bit address needs to be calculated relative to its parent. If that parent is a bank, how that bank modifies the address needs to be calculated first, and then continue working up the bank structure until the memory map is reached. To do so, the following formulas apply.

— For a bit in an address block directly in a memory map:

address_offset = offset in address unit bits *(a)*

The *address_offset* describes the offset in address-unit-bits. In IP-XACT, the following items' offset are described in address-unit-bits: addressBlock base-address, register and register-file addressOffset, bank baseAddress, and subspaceMap baseAddress.

bit_offset = offset in bits *(b)*

The *bit_offset* describes the offset in bits. In IP-XACT, the bitOffset for fields is described in bits.

$$addressBlock_bit_address = ((address_offset^{(a)} + addressBlock.baseAddress) \times memoryMap.addressUnitBits) + bit_offset^{(b)} \quad (1)$$

— For a bit in a subspace map:

$$subspaceMap_bit_address = ((address_offset^{(a)} + subspaceMap.baseAddress) \times addressSpace.addressUnitBits) + bit_offset^{(b)} \quad (2)$$

However, the following formulas need to be used on any containing banks.

a) For an item (bank, subspace map, or address block) within a serial bank:

$$container_addressUnitBits := memoryMap.addressUnitBits \mid addressSpace.addressUnitBits \quad (3)$$

$$item_width := addressBlock_width^{(14)} \mid subspaceMap_width^{(15)} \mid bank_width(^{(16)} \quad (4)$$

$$item_range := addressBlock_range^{(8)} \mid subspaceMap_range^{(9)} \mid bank_range^{(11)} \quad (5)$$

$$item_rows = ceiling((item_range^{(5)} \times container_addressUnitBits^{(3)}) / item_width^{(4)}) \quad (6)$$

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

The effective range of an item is its range rounded up to the nearest complete row:

$$item_effective_range = item_rows^{(6)} \times item_width^{(4)} \quad (7)$$

The range of an item is calculated depending on its type:

1) For an address block the range is the value of the range subelement;

$$addressBlock_range = addressBlock.range \quad (8)$$

2) For a subspace map which references a segment, the range is the value of the segment's range elements; for other subspace maps, the range is the value of the address space's range subelement, then the range is normalized by multiplying it with the address space's address unit bits and then divided by the subspace map's memory map address unit bits;

$$subspaceMap_range = (addressSpace_range^{(10)} \times addressSpace.addressUnitBits) / memoryMap.addressUnitBits \quad (9)$$

$$addressSpace_range := addressSpace.range \mid addressSpace.segment.range \quad (10)$$

3) For a bank the range is dependent on its alignment;

$$bank_range := serial_bank_range^{(12)} \mid parallel_bank_range^{(13)} \quad (11)$$

4) For a serial bank, the range is the sum of the effective ranges of the subitems;

$$serial_bank_range = \sum_{i=0}^{n-1} item_effective_range_{[i]}{}^{(7)} \quad (12)$$

5) For a parallel bank, the range is the (largest `item_rows` of all the subitems) x (`bank_width`/`addressUnitBits`);

$$parallel_bank_range = max(item_rows^{(6)}[0], ..., item_rows^{(6)}[n-1]) \times parallel_bank_width(^{(18)} / container_addressUnitBits^{(3)} \quad (13)$$

(i.e., the effective range of an item is its range rounded up to the nearest complete row)

The width of an item is calculated depending on its type:

6) For an address block, the width is defined as the value of the **width** subelement;

$$addressBlock_width = addressBlock.width \quad (14)$$

7) For a subspace map, the width is the width of the address space of the referenced bus interface;

$$subspaceMap_width = addressSpace.width \quad (15)$$

8) For a serial bank, the width is the width of the widest subitem;

$$bank_width := serial_bank_width^{(17)} \mid parallel_bank_width^{(18)} \quad (16)$$

$$serial_bank_width = max(item_width^{(4)}[0], \ldots, item_width^{(4)}[n-1]) \quad (17)$$

9) For a parallel bank, the width is the sum of the widths of the subitems.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

$$parallel_bank_width = \sum_{i=0}^{n-1} item_width_{[i]}^{(4)} \quad (18)$$

b) For an offset within item *n* in a serial bank:

$$serial_bank_bit_address = bit_offset^{(b)} + \left(\sum_{i=0}^{n-1} item_effective_range_{[i]}^{(7)} \times container_addressUnitBits^{(3)} \right) \quad (19)$$

c) For a bit within item *n* in a parallel bank containing *m* items:

$$bit_offset_in_row = bit_offset^{(b)} \bmod item_width^{(4)}[n] + \sum_{i=0}^{n-1} item_width_{[i]}^{(4)} \quad (20)$$

$$row_bit_offset = \sum_{i=0}^{n-1} item_width_{[i]}^{(4)} \times \left(bit_offset^{(b)} / item_width^{(4)}[n] \right) \quad (21)$$

$$parallel_bank_bit_address = row_bit_offset^{(21)} + bit_offset_in_row^{(20)} \quad (22)$$

Once the bit address within a top-level bank has been calculated, the bit address within the memory map can be derived from the following formula:

$$bank_bit_address := parallel_bank_bit_address^{(22)} \mid serial_bank_bit_address^{(19)} \quad (23)$$

$$memory_map_bit_address = address_space_bit_address = block_bit_address^{(25)} + (item.baseAddress \times container_addressUnitBits^{(3)}) \quad (24)$$

$$block_bit_address := subspaceMap_bit_address^{(2)} \mid addressBlock_bit_address^{(1)} \mid bank_bit_address^{(23)} \quad (25)$$

12.2 Calculating the bus address at the slave bus interface

The bus address of a bit at the slave bus interface can be derived from the following formulas:

$$slave_bus_address = memory_map_bit_address^{(24)} / slave.bitsInLau \quad (26)$$

On a bus, the bit offset gives the offset within the LAU of the bit using the following formula:

$$slave_bit_offset = memory_map_bit_address^{(24)} \bmod slave.bitsInLau \quad (27)$$

12.3 Calculating the address at the indirect interface

The address of a bit at an indirect interface can be derived from the following formula:

$$indirect_interface_bus_address = memory_map_bit_address^{(24)} / indirect_interface.bitsInLau \quad (28)$$

The bit offset gives the offset within the LAU of the bit using the following formula:

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

indirect_interface_bit_offset = memory_map_bit_address$^{(24)}$ *mod indirect_interface.bitsInLau* *(29)*

12.4 Address modifications of a channel

The address at the mirrored slave interface can be derived from the following formula:

mirrored_slave_bus_address = (slave_bus_address$^{(26)}$ x *slave.bitsInLau) / mirroredSlave.bitsInLau* *(30)*

This is then modified by the remap address:

mirrored_slave_row_address = mirrored_slave_bus_address$^{(30)}$ *+ (mirroredSlave.baseAddress.remapAddress)* *(31)*

where *remapAddress* is the remap address for the current state of the channel.

mirrored_master_bus_address = (mirrored_slave_row_address$^{(31)}$ x *mirroredSlave.bitsInLau) / mirroredMaster.bitsInLau* *(32)*

12.5 Addressing in the master

The bus address at the master bus interface can be derived from the following formula:

master_bus_address = (mirrored_master_bus_address$^{(32)}$ x *mirroredMaster.bitsInLau) / master.bitsInLau* *(33)*

This gives a bit address of

master_bit_address = master_bus_address$^{(33)}$ x *master.bitsInLau* *(34)*

The bit address may then be converted to an addressing unit address and offset using the formulas:

addressSpace_bus_address = master_bit_address$^{(34)}$ */ addressSpace.addressUnitBits* *(35)*

addressSpace_bit_offset = master_bit_address$^{(34)}$ *mod addressSpace.addressUnitBits* *(36)*

12.6 Address translation in a bridge

The address at the master interface for a bridge can be derived from the following formulas:

a) The bus address at the master bus interface is:

bridge_master_bus_address = slave_bus_address$^{(26)}$ *(37)*

This gives a bit address of

bridge_master_bit_address = bridge_master_bus_address$^{(37)}$ x *bridge_master.bitsInLau + bridge_master.addressSpaceRef.baseAddress* x *addressSpace.addressUnitBits* *(38)*

The master bit address (also equal to the address space bit address) may be converted to an addressing unit address and offset of the *addressSpace* using the formulas:

bridge_address_space_address = bridge_master_bit_address$^{(38)}$ */ addressSpace.addressUnitBits* *(39)*

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

bridge_address_space_offset = bridge_master_bit_address[38] *mod addressSpace.addressUnitBits* *(40)*

b) The bit address may also be converted to the address of the bridged slave interface by using the following formulas:

1) For a transparent bridge:

bridge_slave_address = bridge_address_space_address[39] x *addressSpace.addressUnitBits / bridge_slave.bitsInLau* *(41)*

2) For an opaque bridge:

bridge_slave_address = (((bridge_address_space_address[39] *– segment.addressOffset)* x *addressSpace.addressUnitBits) / bridge_slave.bitsInLau) + slave_bus_address*[26] *(42)*

c) When an indirect interface is bridging to a master, the bit address may also be converted to the address of the indirect interface using the following formulas:

1) For a transparent bridge:

bridge_indirect_interface_address = (bridge_address_space_address[39] x *addressSpace.addressUnitBits) / bridge_indirect_interface.bitsInLau* *(43)*

2) For an opaque bridge:

bridge_indirect_interface_address = (((bridge_address_space_address[39] *- segment.addressOffset)* x *addressSpace.addressUnitBits) / bridge_indirect_interface.bitsInLau) + slave_bus_address*[26] *(44)*

13. Data visibility

The addressing descriptions in Clause 12 presume each bus interface only maps a single logical address port (a port with an **isAddress** qualifier) and a single logical data port (a port with an **isData** data qualifier). See also: 5.6 and 5.9.

If a bus interface maps more than one address or data port, then each combination of address and data ports implies a separate addressing and data visibility calculation. To calculate the address map for a particular type of transaction, the data and address ports that transaction uses need to be known first.

The most common case for multiple data ports in a single bus interface is where there are separate read and write data ports; however, their relevant properties of the read and write data ports are typically identical—giving identical read and write address maps.

13.1 Mapped address bits mask

The mapped address bits need to be taken into account when calculating the data visibility to the interface by deriving a mask from the set of address bits mapped in the interface. This mask is '`bitwise and`ed' with the `bus_address`.

interface_mapped_address_bits = a mask derived from the set of address bits mapped in the interface (c)

interface_bus_address := slave_bus_address[26] *| mirrored_slave_bus_address*[30] *| mirrored_master_bus_address*[32] *| master_bus_address*[33] *| bridge_master_bus_address*[37] *(45)*

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

interface_visible_bus_address = interface_bus_address$^{(45)}$ *& interface_mapped_address_bits*$^{(c)}$ *(46)*

13.2 Address modifications of an interconnection

The bus address is carried between adjacent bus interfaces (slave and mirrored slave, master and mirrored master, or master and slave) on the bus's **isAddress** logical port. If this port is a wire port, the address is always carried as parallel bits with the least significant bit of the address on logical bit `0` of the port. The interconnection can modify the address in two ways:

a) If some address bits are not connected, addresses with those bits set are not accessible from the master.
   1) Examine the logical vectors in the port maps to determine which address bits are connected.
   2) Transactional ports always carry all address bits across the interconnection.

b) If the value of **bitsInLau** differs on the two sides of the interconnection, the interpretation of the address as a bit address can vary by the ratio of the interfaces' **bitsInLau**. This, however, does not alter the actual bus address.

13.3 Bit steering in a channel

How addresses are modified within a channel depends on the value of **bitSteering** in the mirrored slave interface. It also depends on the relative width of the mirrored master and mirrored slave data ports, where this width is defined to be the total number of bits of the logical data port that are mapped in the bus interface. If `bitSteering` is `on`, or the slave is wider than or the same width as the master, the addresses are simply modified to take into account any change in **bitsInLau** between the mirrored slave and the mirrored master, as shown in the following formula:

mirrored_master_steering_on_visible_bus_address = mirrored_master_bus_address$^{(32)}$ *& mirrored_master_mapped_address_bits*$^{(c)}$ *(47)*

If `bitSteering` is `off` and the mirrored slave is narrower than the mirrored master, the address is adapted so all locations in the slave's memory map are visible:

mirroredMaster_width = relative width of the dataport of the mirrored master interface (d)

mirroredSlave_width = relative width of the dataport of the mirrored slave interface (e)

mirrored_slave_bit_address = mirrored_slave_row_address$^{(31)}$ x *mirroredSlave.bitsInLau (48)*

mirrored_master_bit_address = mirrored_slave_bit_address$^{(48)}$ *mod mirroredSlave_width*$^{(e)}$ + *((mirrored_slave_bit_address*$^{(48)}$ */ mirroredSlave_width*$^{(e)}$*)* x *mirroredMaster_width*$^{(d)}$*) (49)*

mirrored_master_steering_off_visible_bus_address = mirrored_master_bit_address$^{(49)}$ */ mirroredMaster.bitsInLau & mirrored_master_mapped_address_bits*$^{(c)}$ *(50)*

Finally, **bitSteering** has a different meaning in a mirrored slave interface than in a master or slave interface. In a master or slave interface, it means the component shall modify which bit lanes are used for data when accessing narrow devices. In a mirrored slave interface, it means the addresses from a mirrored master interface are not modified for transfers to a narrower mirrored slave data port.

13.4 Visibility of bits

A bit in the slave's memory map is visible in the master's address space if:

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

— it is in an address range visible to the master;

— the master and slave agree on which bit lane the bit should appear and this bit lane is connected between the master and the slave.

13.4.1 Visible address ranges

Two conditions need to be fulfilled for an address in the slave to be visible to the master.

a) The address at the mirrored slave shall be within the range supported by the mirrored slave interface:

$$mirrored_slave_visible_bus_address < mirroredSlave.baseAddress.range \quad (51)$$

b) The address in the address space shall be within the range supported by the master address space for that bus interface:

$$0 <= master_bit_address^{(34)} < addressSpace.range \times addressSpace.addressUnitBits \quad (52)$$

13.4.2 Bit lanes in memory maps

The local bit lane of a bit in an address block is:

$$address_block_bit_lane = addressBlock.bit_offset^{(b)} \bmod addressBlock.width \quad (53)$$

Similarly, in a subspace map the bit lane is:

$$subspace_map_bit_lane = subspaceMap.bit_offset^{(b)} \bmod addressSpace.width \quad (54)$$

where the *addressSpace.width* is the width of the address space of the referenced master bus interface.

If the address block or subspace map is at the top-level of the memory map or only within serial banks, the bit lane in the memory map is the local bit lane.

$$local_bit_lane := address_block_bit_lane^{(53)} \mid subspace_map_bit_lane^{(54)} \quad (55)$$

If it is item *n* in a parallel bank, then:

$$bank_bit_lane = \sum_{i=0}^{n-1} item_width_{[i]}{}^{(4)} + local_bit_lane^{(55)} \quad (56)$$

If it is in multiple hierarchical parallel banks, this formula is applied at each higher level with the lower-level *bank_bit_lane* replacing *local_bit_lane*.

The bit lane in the memory map is the top-level *bank_bit_lane*.

13.4.3 Bit lanes in address spaces

The bit lane in an address space can be derived from the following formula:

$$address_space_bit_address = master_bit_address^{(34)} \quad (57)$$

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

address_space_bit_lane = address_space_bit_address[(57)] *mod addressSpace.width* *(58)*

13.4.4 Bit lanes in bus interfaces

In a bus interface, the logical bit numbers of the data port carry the corresponding bit lanes. For example, if a slave bus interface has a data port with a logical vector of `[15:8]`, this port can access bit lanes `15` to `8` of the memory map and logical bit lanes `15` to `8` in the connected mirrored slave or master interface.

13.4.5 Bit lanes in channels

All bus interfaces on a channel shall use the same logical numbering of data port bits. This means data bits cannot be moved between bit lanes in a channel by giving the mirrored bus interfaces different logical to physical mappings on their data ports.

13.4.6 Bit steering in masters and slaves

Bit steering only takes effect when the master and the slave have data ports of different widths. If they do and bit steering is enabled (i.e., `bitSteering` is `on` in the master or slave interface) for the bus interface with the wider data port, then this data port shall move its copy of output data to the correct bit lanes for the narrower port and read its input data from the correct bit lanes for the narrower port.

If bit steering is disabled in the wider port, the master can only access data at a particular address when the bit lane for that address in the address space is connected (through the bus interfaces and a channel) to the bit lane for the corresponding address in the memory map.

The following also apply.

— The **bitSteering** value has a different meaning in mirrored slaves. See 12.2.

— Some buses with bit steering may only support certain data port widths. Describing which widths are supported is outside the scope of IP-XACT.

— Bit steering allows software or hardware away from the bus interface to work without knowing the width of devices on the far side of the bus. To provide this functionality, a bus supporting bit steering normally gives the same address bits to all devices, irrespective of their widths, and does not adapt addresses to the width of the slave bus interfaces (i.e., `bitSteering` is `on` in the mirrored slave bus interfaces). Thus, a non-bitsteering master on such a bus only has access to some of the memory rows of narrower slaves.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Annex A

(informative)

Bibliography

[B1] IEC/IEEE 61691-1-1, Behavioral languages—Part 1: VHDL language reference manual.[16, 17]

[B2] IEEE Std 754™-1985, IEEE Standard for Binary Floating-Point Arithmetic.[18]

[B3] IEEE Std 1364™, IEEE Standard for Verilog Hardware Description Language.

[B4] IEEE Std 1666™, IEEE Standard for SystemC Language Reference Manual.

[B5] IETF RFC 2119, "Key words for use in RFCs to Indicate Requirement Levels," Bradner, S., Best Current Practice: 14.[19]

[B6] IP-XACT Leon Register Transfer Examples.[20]

[B7] IP-XACT Leon Transaction Level Examples.[21]

[B8] IP-XACT Schema Examples.[22]

[B9] IP-XACT Schema.[23]

[B10] IP-XACT Schema on-line documentation.[24]

[B11] IP-XACT TGI WSDL.[25]

[B12] ISO/IEC 8859-1, Information technology—8-bit single-byte coded graphic character sets—Part 1: Latin Alphabet No. 1.[26]

[B13] ISO/IEC 8879, Information processing—Text and office systems—Standard Generalized Markup Language (SGML).

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

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

[18]The IEEE standards or products referred to in this annex are trademarks of the Institute of Electrical and Electronics Engineers, Inc.

[19] Available from http://www.ietf.org/rfc/rfc2119.txt.

[20]Available from http://www.accellera.org/doc_downloads/.

[21]Available from http://www.accellera.org/XMLSchema/1685-2009/index.xsd.

[22]See Footnote 20.

[23]See Footnote 21.

[24]See Footnote 21.

[25]Available from http://www.accellera.org/XMLSchema/1685-2009/TGI/TGI.wsdl.

[26]ISO/IEC publications are available from the ISO Central Secretariat, Case Postale 56, 1 rue de Varembé, CH-1211, Genève 20, Switzerland/Suisse (http://www.iso.ch/). ISO/IEC publications are also available in the United States from Global Engineering Documents, 15 Inverness Way East, Englewood, CO 80112, USA (http://global.ihs.com/). Electronic copies are available in the United States from the American National Standards Institute, 25 West 43rd Street, 4th Floor, New York, NY 10036, USA (http://www.ansi.org/).

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

[B14] International System of Units.[27]

[B15] IP-XACT standard tool names for **envIdentifier**.[28]

[27]Available at http://physics.nist.gov/cuu/Units/.
[28]Available at http://www.accellera.org/tech/refs/toolnames.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Annex B

(normative)

Semantic consistency rules

For an IP-XACT document or a set of IP-XACT documents to be valid, they shall, in addition to conforming to the IP-XACT schema, obey certain semantic rules. While many of these are described informally in other clauses of this document, this annex defines them formally. Tools generating IP-XACT documents shall ensure these rules are obeyed. Tools reading IP-XACT documents shall report any breaches of these rules to the user.

B.1 Semantic consistency rule definitions

The following definitions apply when determining a semantic consistency rule (SCR) interpretation.

B.1.1 Compatibility of busDefinitions

A set of **busDefinition**s are considered compatible if they are related by the **extends** relationship as described in 5.11.1. A **busDefinition** is always compatible with itself.

B.1.2 Interface mode of a bus interface

Specifies which of the following exclusive sub-elements of the **busInterface** is *present*: **master**, **slave**, **system**, **mirroredMaster**, **mirroredSlave**, **mirroredSystem**, or **monitor**. For instance, a **busInterface** containing a system sub-element is considered to have an interface mode of **system**.

B.1.3 Compatibility of abstractionDefinitions

A set of **abstractionDefinition**s are considered compatible if they are related by the *extending* relationship as described in 5.11.2. Two configured **abstractionDefinition**s are compatible with each other if they meet the *extending* requirements.

B.1.4 Element referenced by configurableElementValue element

A **configurableElementValue** element is considered to reference a configurable element if the value of the **referenceId** attribute of the **configurableElementValue** matches the **parameterId** of that configurable element.

B.1.5 Memory mapping

If an access is not specified, the value defaults to the value from the level above. If the top level is not specified, the access defaults to read-write.

B.1.6 Port connection equivalence class

The *port connection equivalence class* of a (logical or component) port is the set of model and logical ports that can be reached from that port through any sequence of the following:

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

a) Bus interfaces' logical to physical port maps
b) Interconnections between logical ports implied by interconnections between bus interfaces using the same abstraction of the bus
c) Ad hoc connections

B.1.7 Logical and physical ports

a) If a **physicalPort** has a **partSelect** subelement, the physical bits specified by the **partSelect** are ordered using System Verilog bit ordering rules. If there is no **partSelect** subelement, the bit ordering is defined by the **arrays** subelements (if any) followed by the **vectors** subelements (if any) of the model port, again using System Verilog bit ordering rules. This bit ordering defines a linear order of the specified bits from `physical.left` to `physical.right`. If the model port does not have a range defined, then `physical.left` = `physical.right = 0`.
b) If a **logicalPort** element has a **range** subelement, its range shall be [**left**:**right**], where **left** and **right** are the left and right values of the **range** subelement. If a **logicalPort** element does not have a **range** subelement and the **physicalPort** element in the same **portMap** references a wire port, then its range shall be taken as [`abs(physical.left – physical.right):0`].
c) A logical bit of a port is mapped if it is included in the range of a **logicalPort** element referencing that bus interface port.

B.1.8 Addressable bus interface

A bus interface shall be *addressable* if the **isAddressable** element of the bus definition it references has the value **true**.

B.2 Rule listings

The semantic rules listed here can be checked purely by manually examining a set of IP-XACT documents. In Table B.1 through Table B.16, *Single doc check* indicates a rule can be checked purely by manually examining a single IP-XACT document. Rules for which *Single doc check* is `No` require the examination of the relationships between IP-XACT documents. When *Post config* is `Yes`, that rule only applies after configuration has been completed.

NOTE—Where these tables contain references to the values of elements and those elements are configurable in IP-XACT, the values used are the configured values (not the XML element values).

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

B.2.1 Cross-references and VLNVs

Table B.1—Cross-references and VLNVs

Name	Rule	Single doc check	Post config
SCR 1.1 *uniqueVLNV*	Every IP-XACT document visible to a tool shall have a unique VLNV. Only applies to those documents visible to a particular tool or DE at one time. In particular, users are likely to store multiple versions of the same documents, with the same VLNVs, in source control systems. See also: C.25.2 and C.25.4.	No	No
SCR 1.2 *anyVLNVRefMustExist*	Any VLNV in an IP-XACT document used to reference another IP-XACT document shall precisely match the identifying VLNV of an existing IP-XACT document. In the schema, such references always use the attribute group **versionedIdentifier**. See also: C.25.2 and C.25.4.	No	No
SCR 1.3 *busDefExtendsVLNVIsBusDef*	The VLNV in an **extends** element in a bus definition shall be a reference to a bus definition. See also: 5.2.2.	No	No
SCR 1.4 *busTypeVLNVIsBusDef*	The VLNV in a **busType** element in a bus interface or abstraction definition shall be a reference to a bus definition. See also: 6.5.1.	No	No
SCR 1.5 *designRefVLNVIsDesign*	The VLNV in a **designRef** element in a design configuration or **designInstantiation** element in a component shall be a reference to a **design**. See also: 10.2.2.	No	No
SCR 1.6 *designCfgRefVLNVisDesignCfg*	The VLNV in a **designConfigurationRef** element in a **designConfigurationInstantiation** element in a component shall be a reference to a **designConfiguration**.	No	No
SCR 1.7 *cfgChainRefVLNVIsGenChain*	The VLNV in a **generatorChainRef** element in a design configuration shall be a reference to a generator chain. See also: 10.2.2 and 10.3.	No	No
SCR 1.8 *selChainRefVLNVIsGenChain*	The VLNV in a **generatorChainRef** subelement of the element **generatorChainSelector** in a generator chain shall be a reference to a generator chain. See also: 9.2.2.	No	No
SCR 1.9 *compRefVLNVisComp*	The VLNV in a **componentRef** element in a **design** shall be a reference to a **component**. See also: 7.2.2.	No	No
SCR 1.10 *legalTopDocTypes*	The XML document element of an IP-XACT document shall be an **abstractor**, **abstractionDefinition**, **busDefinition**, **component**, **design**, **designConfiguration**, **generatorChain**, or **catalog** element. See also: 5.2.2, 5.3, 6.1.2, 7.1.2, 8.1.2, 9.1.2, and 10.2.2.	Yes	No
SCR 1.11 *absTypeVLNVIsAbsDef*	The VLNV in an **abstractionType** element in a component or abstractor shall reference an **abstractionDefiniton**. See also: 8.1.2.	No	No

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table B.1—Cross-references and VLNVs *(continued)*

Name	Rule	Single doc check	Post config
SCR 1.12 *busTypeMustMatch*	If a bus interface contains an **abstractionType** subelement, the abstraction definition's **busType** element and the bus interface's **busType** element shall reference the same bus definition. I.e., the abstraction referenced shall be an abstraction of the referenced bus. See also: 5.2.2, 5.3.2, and 6.5.1.2.	No	No
SCR 1.13 *absRefVLNVIsAbstractor*	The VLNV in an **abstractorRef** in a **designConfiguration** shall reference an **abstractor**. See also: 10.3.2.	No	No
SCR 1.14 *absDefExtendsVLNVIsAbsDef*	The VLNV in an **extends** element in an abstraction definition shall be a reference to an abstraction definition. See also: 5.3.2.	No	No
SCR 1.15 *CatalogVLNVReference*	The VLNV within an **ipxactFile**/**vlnv** element in a catalog needs to refer to an IP-XACT file whose file type matches the container of the **ipxactFile**/**vlnv** element. E.g., all VLNV's within the **catalog busDefinitions** element shall refer to IP-XACT **busDefinitions** elements. See also: 11.2.2.	No	No
SCR 1.16 *viewDesignConfigurationInstantiationReference*	If a **view** contains a **designConfigurationInstantiationRef** and does not also contain a designInstantiationRef, then the referenced design configuration document shall contain a **designRef** element. See also: 6.12.1.2 and 10.2.2.	Yes	Yes
SCR 1.17 *DesignRefsMatch*	If a **view** contains a **designConfigurationInstantiationRef** and a **designInstantiationRef**, and the design configuration contains a **designRef**, then the referenced design VLNVs shall match. See also: 6.12.1.2 and 10.2.2.	No	Yes

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

B.2.2 Interconnections

Table B.2—Interconnections

Name	Rule	Single doc check	Post config
SCR 2.1 *connectedIntfsMustExist*	In the attributes of an **activeInterface**, **monitoredActiveInterface** or **monitorInterface** element, the value of the **busRef** attribute shall be the name of a **busInterface** in the component description referenced by the VLNV of the component instance named in **componentRef** and optional **path** attributes. See also: 7.3.1.2 and 7.4.2.	No	No
SCR 2.2 *connectedIntfsCompat*	In the subelements of an **interconnection**, the bus interfaces referenced by all **activeInterface** and **hierInterface** subelements shall be compatible, i.e., the **busType** elements within the **busInterface** elements shall reference compatible **busDefinitions**. See also: 6.3.1, 6.3.2, 6.3.3, and 7.3.1.2.	No	Yes
SCR 2.3 *intfConnectedOnlyOnce*	A particular component/bus interface combination shall appear in only one **interconnection** element in a design. See also: 7.3.1.2.	Yes	Yes
SCR 2.4 *activeMstConnect*	An active interface of type **master** shall only connect to active interfaces of type **slave** or **mirrored-master**, or hierarchical interfaces of type **master**. See also: 7.3.1.2.	No	No
SCR 2.5 *activeMSlvConnect*	An active interface of type **mirrored-slave** shall only connect to active interfaces of type **slave** or hierarchical interfaces of type **mirrored-slave**. See also: 7.3.1.2.	No	No
SCR 2.6 *hierSlvConnect*	A hierarchical interface of type **slave** shall only connect to active interfaces of type **slave**. See also: 7.3.1.2.	No	No
SCR 2.7 *hierMMstConnect*	A hierarchical interface of type **mirrored-master** shall only connect to active interfaces of type **mirrored-master**. See also: 7.3.1.2.	No	No
SCR 2.8 *systemConnect*	An active interface of type **system** shall only connect to active interfaces of type **mirrored-system** or hierarchical interfaces of type **system**. See also: 7.3.1.2.	No	Yes
SCR 2.9 *MstSystemConnect*	An active interface of type **mirrored-system** shall only connect to active interfaces of type **system** or hierarchical interfaces of type **mirrored-system**. See also: 7.3.1.2.	No	Yes
SCR 2.10 *interconnectionDriver*	An **interconnection** element without system/mirrored-system interfaces shall contain one driving interface, which is an active interface referencing a master, a mirrored-slave, or a hierarchical interface referencing a slave or mirrored-master. See also: 7.3.1.2.	No	Yes

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table B.2—Interconnections *(continued)*

Name	Rule	Single doc check	Post config
SCR 2.11 *MstToSlvBitsLAUMatch*	In a direct master to slave connection, the value of **bitsInLAU** in the master's bus interface shall match the value of **bitsInLAU** in the slave's bus interface. See also: 6.3.1 and 7.3.1.2.	No	No
SCR 2.12 *MstToSlvIsDirectConnect*	In a direct master to slave connection, the **bus-Definitions** referenced by the **busInterfaces** shall have a **directConnection** element with the value **true**. See also: 6.3.1 and 7.3.1.2.	No	No
SCR 2.13 *SysToMSysGroupsMatch*	In a connection between a system interface and a mirrored-system interface, the values of the **group** elements of the two bus interfaces shall be identical. See also: 6.3.1, 6.3.2, 6.5.2.2, and 7.3.1.2.	No	No
SCR 2.14 *EndianessMustMatch*	The endianess in all bus interfaces shall match for any interconnection using an addressable bus. If the endianess is not specified at either bus interface, it is presumed to be little endian. See also: 6.3.1, 6.3.2, 6.5.1.2, and 7.3.1.2.	No	No
SCR 2.15 *ConnectionRequired*	If a design contains a component with a **busInterface** that has a **connectionRequired** element with the value **true**, that **busInterface** shall be included in an **interconnection** of the design. See also: 6.5.1.2 and 7.3.1.2.	No	No
SCR 2.16 *MonIntfPathMustExist*	A **monitorInterconnection** with interfaces that contain a **path** attribute with a **componetRef** and **busRef** shall exist in all hierarchical views. See also: 7.4.	No	Yes
SCR 2.17 *broadcastConstraint*	An interconnection may not contain more than two total **activeInterface** and **hierInterface** elements unless the underlying bus definition has the **broadcast** element set to **true**. See also: 5.2.2 and 7.3.1.2.	Yes	Yes
SCR 2.18 *excludePortExists*	A physical port name referenced in an **excludePort** element shall match the name of a **port** defined in the ports list of the **component**. See also: 7.4.2.	No	No

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

B.2.3 Channels, bridges, and abstractors

Table B.3—Channels, bridges, and abstractors

Name	Rule	Single doc check	Post config
SCR 3.1 *ChannelAbsDefsCompat*	Within a **channel** element, all the **busInterfaceRef** elements shall refer to compatible abstraction definitions, i.e., the VLNVs of the **abstractionType** elements within the **busInterface** elements shall reference compatible **abstractionDefinition**s. Compatibility of the abstraction definitions implies compatibility of their associated bus definitions. See also: 5.3.2 and 6.7.2.	No	Yes
SCR 3.2 *ChannelIntfsMirrored*	All bus interfaces referenced by a channel shall be mirrored interfaces. See also: 6.4.1 and 6.7.2.	Yes	No
SCR 3.3 *MaxMastersHonored*	A channel can be connected to no more mirrored-master **busInterface**s than the least value of **maxMasters** in the bus definitions referenced by the connected bus interfaces (whether these interfaces are mirrored-master or mirrored-slave interfaces). A channel may connect ports with different bus definitions, and hence different values of **maxMasters**, as long as the bus definitions are compatible. See also: 6.7.2.	No	Yes
SCR 3.4 *MaxSlavesHonored*	A channel can be connected to no more mirrored-slave bus interfaces than the least value of **maxSlaves** in the bus definitions referenced by the connected bus interfaces (whether these interfaces are mirrored-master or mirrored-slave interfaces). A channel may connect ports with different bus definitions, and hence different values of **maxSlaves**, as long as the bus definitions are compatible. See also: 6.7.2.	No	Yes
SCR 3.5 *BusIntfInOneChannelMax*	Each bus interface on a component shall connect to only one channel of that channel component. See also: 6.7.2.	Yes	Yes
SCR 3.6 *MasterRefIntfIsMaster*	The interface referenced by **masterRef** subelement of a **bridge** element shall be a master. See also: 6.5.4.2.	Yes	No
SCR 3.7 *InterConnectionRefMustMatch*	The value of the **interconnectionRef** subelement of an **interconnectionConfiguration** element shall precisely match a design **interconnection**/**name** or a design **monitorInterconnection**/**name** of an interconnection described in the design referenced by the containing design configuration or referenced by the design in the **designInstantiation** referenced by the view referencing the containing design configuration. See also: 10.3.2.	No	No

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table B.3—Channels, bridges, and abstractors *(continued)*

Name	Rule	Single doc check	Post config
SCR 3.8 *AbstractorModeMustBeMaster*	An **abstractors** element of an **interconnectionConfiguration** element in a design configuration document that references a master to mirrored master connection shall only reference **abstractors** with an **abstractorMode** of **master**. See also: 10.3.2.	No	No
SCR 3.9 *AbstractorModeMustBeSlave*	An **abstractors** element of an **interconnectionConfiguration** element in a design configuration document that references a slave to mirrored-slave interconnection in the corresponding design shall only reference **abstractors** with an **abstractorMode** of **slave**. See also: 10.3.2.	No	No
SCR 3.10 *AbstractorModeMustBeSystem*	An **abstractors** element of an **interconnectionConfiguration** element in a design configuration document that references a system to mirrored-system interconnection in the corresponding design shall only reference **abstractors** with an **abstractorMode** of **system**. See also: 10.3.2.	No	No
SCR 3.11 *AbstractModeMustBeDirect*	An **abstractors** element of an **interconnectionConfiguration** element in a design configuration document that references a master to slave interconnection in the corresponding design shall only reference **abstractors** with an **abstractorMode** of **direct**. See also: 10.3.2.	No	No
SCR 3.12 *AbstractionChainStart*	In the list of **abstractor** elements within an **abstractors** element in an **interconnectionConfiguration** element, the first **abstractionType** element of the first referenced **abstractor** shall be compatible with the **abstractionType** element of the master, system, or mirrored-slave endpoint of the interconnection.[a] See also: 10.3.2.	No	No
SCR 3.13 *AbstractionChainEnd*	In the list of **abstractor** elements within an **abstractors** element an **interconnectionConfiguration** element, the second **abstractionType** element of the last referenced abstractor shall be compatible with the **abstractionType** element of the mirrored-master, mirrored-system, or slave endpoint of the interconnection.[a] See also: 10.3.2.	No	No
SCR 3.14 *AbstractionChainMiddle*	In the list of **abstractor** elements within an **abstractors** element an **interconnectionConfiguration** element, the first **abstractionType** element of every referenced abstractor, except the first, shall be compatible with the second **abstractionType** element of the previous abstractor in the **interconnection-Configuration** list.[a] See also: 10.3.2.	No	No
SCR 3.15 *AbstractionBusTypesMust Match*	The VLNVs in the **busType** elements of both abstraction definitions referenced by an abstractor shall exactly match the VLNV in the **busType** element of the abstractor. See also: 5.3.2 and 8.1.2.	No	No

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table B.3—Channels, bridges, and abstractors *(continued)*

Name	Rule	Single doc check	Post config
SCR 3.16 *AbstractionExtendsCondition*	If abstraction definition AA is an abstraction of bus definition A and abstraction definition AB is an abstraction of bus definition B, then abstraction definition AA shall only contain an **extends** element referencing abstraction definition AB if bus definition A contains an **extends** element referencing bus definition B. If abstraction definition AA extends abstraction definition AB, AA and AB need to be abstractions of different buses. See also: 5.3.2.	No	No
SCR 3.17 *SubspaceMasterRefExists*	The interface referenced by the **masterRef** attribute of a **subspaceMap** element shall be a master interface. See also: 6.9.9.2.	Yes	No
SCR 3.18 *multiAbstractorBroadcast*	If multiple **abstractors** elements appear in an **interconnectionConfiguration**, then the referenced **interconnection** shall be a broadcast connection (i.e., contain more than two interfaces). See also: 5.2.2 and 10.3.2.	Yes	No

[a]SCR 3.12 – SCR 3.14 mean the abstractors associated with an interconnection need to form a non-looping chain between the two ends.

B.2.4 Monitor interfaces and monitor interconnections

Table B.4—Monitor interfaces and monitor interconnections

Name	Rule	Single doc check	Post config
SCR 4.1 *ActiveInterfaceCondition*	An **activeInterface** or **monitoredActiveInterface** element shall reference a **master**, **slave**, **system**, **mirroredMaster**, **mirroredSlave**, or **mirroredSystem** interface. See also: 6.3.3, 7.3.1.2, 7.4.2, and 7.6.2.	No	No
SCR 4.2 *MonitorInterfaceCondition*	The **monitorInterface** subelements of a **monitorInterconnection** element shall reference a **monitor** bus interface. See also: 6.3.3 and 7.3.1.2.	No	No
SCR 4.3 *MonitorModeMustMatch*	In a **monitorInterconnection** element, the value of the **interfaceMode** of the monitor interfaces shall match the mode of the **monitoredActiveInterface**. This means all the monitor interfaces shall have the same interface mode. See also: 6.3.3, 6.5.2.2, and 7.3.1.2.	No	No
SCR 4.4 *MonitorSystemGroupMatches*	A **monitor** interface shall only be connected to a **system** or **mirroredSystem** interface if it has a **group** subelement and the value of this element matches the value of the **group** subelement of the **system** or **mirroredSystem** interface. See also: 6.3.3, 6.5.2.2, and 7.3.1.2.	No	No

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table B.4—Monitor interfaces and monitor interconnections *(continued)*

Name	Rule	Single doc check	Post config
SCR 4.5 *InterfaceAppearsOnce*	A particular **componentRef**/**busRef** combination shall only appear in one **monitorInterconnection** element. This applies to both monitor and active interfaces; however, a single **monitorInterconnection** element can connect an active interface to many monitor interfaces. The same active interface can also appear in at most one **interconnection** element. See also: 6.3.3 and 7.3.1.2.	No	Yes
SCR 4.6 *MonitorPortDirRequirement*	All ports mapped in a **busInterface** with a mode of monitor shall have a direction of **in** for **wire** type ports or **provides** for **transactional** type ports. See also: 6.3.3	Yes	No

B.2.5 Configurable elements

Table B.5—Configurable elements

Name	Rule	Single doc check	Post config
SCR 5.1 *expressionFieldValue*	The value of a field that allows expressions shall be an expression and shall only reference **parameters** in the document using their **parameterId**. See also: C.19.2.	Yes	No
SCR 5.2 *parameterIdRequired*	An **parameterId** attribute is required in any element with a **resolve** attribute value of **user** or **generated**. See also: 6.12.6.2 and C.18.2.	Yes	No
SCR 5.3 *componentInstanceConfigurableElementReferences*	**configurableElementValue** elements within **componentInstance** elements shall only reference configurable elements that exist in the **component** referenced by the enclosing **componentInstance** element, excluding those in a **componentInstantiation** or **designConfigurationInstantiation**. The value of the **referenceId** attribute of the **configurableElementValue** element shall match the value of the **parameterId** attribute of some configurable element of the **component**. See also: 7.2.2.	No	No
SCR 5.4 *configElementRefCondition*	**configurableElementValue** elements shall only reference configurable elements. See also: C.5.2.	No	No
SCR 5.5 *configurableElementMin*	If a **configurableElementValue** element references an element with a **type** attribute which does not specify a **string** and contains a **minimum** attribute, the value of the **configurableElementValue** element shall be greater or equal to the specified value of the **minimum** attribute. See also: C.5.2.	No	No

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table B.5—Configurable elements *(continued)*

Name	Rule	Single doc check	Post config
SCR 5.6 *configurableElementMax*	If a **configurableElementValue** element references an element with a **type** attribute which does not specify a **string** and contains a **maximum** attribute, the value of the **configurableElementValue** subelement shall be less than or equal to the specified value of the **maximum** attribute. See also: C.5.2.	No	No
SCR 5.7 *ConfigElementChoiceExists*	If a **configurableElementValue** element references an element with a **choiceRef** attribute, the value for **configurableElementValue** subelement shall be one of the values listed in the **choice** element referenced by the **choiceRef** attribute. See also: 6.14.2 and C.5.2.	No	No
SCR 5.8 *designConfigurationInstantiationConfigurableElementReferences*	**configurableElementValue** elements within **designConfigurationInstantiation** elements shall only reference configurable elements that exist in the **designConfiguration** referenced by the enclosing **designConfigurationInstantiation** element; the value of the **parameterId** attribute of the **configurableElementValue** element shall match the value of the **parameterId** attribute of some configurable element of the design configuration. See also: 6.12.1.2 and C.5.2.	No	No
SCR 5.9 *viewConfigurationConfigurableElementReferences*	**configurableElementValue** elements within **viewConfiguration** elements shall only reference configurable elements that exist in the **componentInstantiation** or **designConfigurationInstantiation** referenced in the component view referenced by the enclosing **view** element; the value of the **referenceId** attribute of the **configurableElementValue** element shall match the value of the parameterId attribute of some configurable element of the component. See also: 10.5.2 and C.5.2.	No	No
SCR 5.10 *generatorChainConfigurableElementReferences*	**configurableElementValue** elements within **generatorChainConfiguration** elements in design configuration documents elements shall only reference configurable elements that exist in the generator chain referenced by the **generatorChainRef** element. See also: 10.2.2 and C.5.2.	No	No
SCR 5.11 *abstractorConfigurableElementReferences*	**configurableElementValue** elements within **interconnectionConfiguration** elements shall only reference **configurable** elements that exist in the **abstractor** referenced by the enclosing **abstractorRef** element. See also: 10.3.2 and C.5.2.	No	No
SCR 5.12 *parameterImplicitCast*	A parameter's value or a configurable element's value shall be implicitly cast to the type specified by the **type** attribute, resulting in an error when the value cannot be cast to the specified type. e.g., **string** to any other types and any other types to **string**. See also: 7.2.	No	No
SCR 5.13 *expressionsMinMax*	Expressions are bound by the values specified by the **minimum** and **maximum** attributes. See also: C.3.	Yes	No

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table B.5—Configurable elements *(continued)*

Name	Rule	Single doc check	Post config
SCR 5.14 *componentInstantiationParameterReferences*	Parameters inside a **componentInstantiation** cannot be referenced by expressions outside that **componentInstantiation**. See also: 6.12.2.2.	Yes	No
SCR 5.15 *designConfigurationInstantiationParameterReferences*	Parameters inside a **designConfigurationInstantiation** cannot be referenced by expressions outside that **designConfigurationInstantiation**. See also: 6.12.2.2.	Yes	No
SCR 5.16 *parameterInitialiasations*	The value of a parameter shall resolve to its indicated type. See also: C.18.	Yes	No
SCR 5.17 *expressionSyntax*	Expressions shall follow the SystemVerilog syntax (see Annex E).	Yes	No
SCR 5.18 *expressionIDsExist*	Any ID used in an expression shall reference an existing **parameterID**.	Yes	No
SCR 5.19 *expressionNonCircular*	Evaluation of an expression shall not lead to circular referencing.	Yes	No
SCR 5.20 *vectorDeclaration*	Vectors shall only be specified on parameters with a type of **bit**. See also: C.18.	Yes	No
SCR 5.21 *arrayDeclaration*	Array parameters shall be fully initialized; it shall be an error if the size of the array as determined by the default value differs from the size the array specified by the **arrays** elements. See also: C.18.	Yes	No
SCR 5.22 *arrayConfiguration*	Arrays shall be fully overridden when configured; it shall be an error if the size of the array resulting from the values specified in the configurable element differs from the size of the array specified by the **arrays** elements. See also: C.18.	Yes	No
SCR 5.23 *designInstantiationConfigurableElementReferences*	**configurableElementValue** elements within **designInstantiation** elements in **component** documents elements shall only reference configurable elements that exist in the design referenced by the enclosing **designInstantiation** element. See also: 6.12.2.2.	No	No
SCR 5.24 *busTypeConfigurable ElementReferences*	**configurableElementValue** elements within **busType** elements shall only reference configurable elements that exist in the bus definition referenced by the enclosing **busType** element. See also: 6.5.1.2.	No	No
SCR 5.25 *abstractionTypeConfigurableElementReferences*	**configurableElementValue** elements within **abstractionType** elements shall only reference configurable elements that exist in the abstraction definition referenced by the enclosing **abstractionType** element. See also: 6.5.1.2.	No	No
SCR 5.26 *assertionValidity*	The value of an assertion shall evaluate to *True*. See also: C.2.	No	Yes

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

B.2.6 Ports

Table B.6—Ports

Name	Rule	Single doc check	Post config
SCR 6.1 *LogicalPortNameExists*	The value of the **name** subelement of any **logicalPort** element within an abstractionType element shall match the value of a **logicalName** element of the abstraction definition referenced by the abstractionType element. See also: 6.5.7.2.	No	No
SCR 6.2 *LogPortRequiresPortDir*	If the abstraction definition referenced by an abstractionType specifies an initiative value for a logical transactional port of **requires** for that interface mode of bus interface, the port map shall only map that logical port to a component port with an initiative value of **requires**, **both**, or **phantom**, or to a component port with an **allLogicalInitiativesAllowed** attribute with the value **true**. For system interfaces, the port initiative values shall be looked up from the **onSystem** element with the group name matching that of the abstractionType's abstraction-definition. For mirrored interfaces, the bus port initiative values needs to be reversed before doing the comparison. See also: 5.10.2, 6.5.7.2, and 6.12.17.2.	No	No
SCR 6.3 *LogPortProvidesPortDir*	If the abstraction definition referenced by an abstractionType specifies an initiative value for a logical transactional port of **provides** for that interface mode of bus or abstractor interface, the port map shall only map that logical port to a component port with an initiative value of **provides**, **both**, or **phantom**, or to a component port with an **allLogicalInitiativesAllowed** attribute with the value **true**. For system interfaces, the port initiative values shall be looked up from the **onSystem** element with the group name matching that of the abstractionType's abstraction-definition. For mirrored interfaces, the bus port initiative values shall be reversed before doing the comparison. Mirrored bus and abstractor interface shall be looked up as if they were not mirrored. See also: 5.10.2, 6.5.7.2, and 6.12.17.2.	No	No

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table B.6—Ports *(continued)*

Name	Rule	Single doc check	Post config
SCR 6.4 *LogPortBothPortDir*	If the abstraction definition referenced by an abstractionType specifies an initiative value for a logical transactional port of **both** for that interface mode of the bus or abstraction interface, and the bus interface has a port map, the port map shall only map that logical port to a component port with an initiative value of **both** or **phantom**, or to a component port with an **allLogicalInitiativesAllowed** attribute with the value **true**. For system interfaces, the port initiative values shall be looked up from the **onSystem** element with the group name matching that of the abstractionType's abstraction-definition. For mirrored interfaces, the bus port initiative values shall be reversed before doing the comparison. Mirrored bus and abstraction interfaces shall be looked up as if they were not mirrored. See also: 5.10.2, 6.5.7.2, and 6.12.17.2.	No	No
SCR 6.5 *LogPortInPortDir*	If the abstraction definition referenced by an abstractionType specifies a direction for a logical wire port of **in** for that interface mode of bus interface, the port map shall only map that logical port to a component port with a direction of **in**, **inout**, or **phantom**, or to a component port with an **allLogicalDirectionsAllowed** attribute with the value **true.** For system interfaces, the port directions shall be looked up from the **onSystem** element with the group name matching that of the abstractionType's abstraction-definition. For mirrored interfaces, the bus port directions shall be reversed before doing the comparison. See also: 5.7.2, 6.5.7.2, 6.12.8.2, and 8.6.2.	No	No
SCR 6.6 *LogPortOutPortDir*	If the abstraction definition referenced by an abstractionType specifies a direction for a logical wire port of **out** for that interface mode of bus interface, the port map shall only map that logical port to a component port with a direction of **out**, **inout**, or **phantom**, or to a component port with an **allLogicalDirectionsAllowed** attribute with the value **true.** For system interfaces, the port directions shall be looked up from the **onSystem** element with the group name matching that of the abstractionType's abstraction-definition. For mirrored interfaces, the bus port directions shall be reversed before doing the comparison. See also: 5.7.2, 6.5.7.2, 6.12.8.2, and 8.6.2.	No	No

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table B.6—Ports *(continued)*

Name	Rule	Single doc check	Post config
SCR 6.7 *LogPortInoutPortDir*	If the abstraction definition referenced by an abstractionType specifies a direction for a logical wire port of **inout** for that interface mode of bus interface, the port map shall only map that logical port to a component port with a direction of **inout** or **phantom**, or to a component port with an **allLogicalDirectionsAllowed** attribute with the value **true.** For system interfaces, the port directions shall be looked up from the **onSystem** element with the group name matching that of the abstractionType's abstraction-definition. For mirrored interfaces, the bus port directions shall be reversed before doing the comparison. See also: 5.7.2, 6.5.7.2, 6.12.8.2, and 8.6.2.	No	No
SCR 6.8 *LogPortPresence*	If the abstraction definition referenced by an abstraction-Type specifies, for a port, a presence value of **required** for that interface mode of bus interface, and the bus interface has a port map, the port shall be in that port map. For system interfaces, the port presence shall be looked up from the **onSystem** element with the group name matching that of the a**bstractionType**'s abstraction-definition. Mirrored bus interfaces shall be looked up as if they were not mirrored. Port maps are optional, even on buses with required ports. See also SCR 6.17. The third possible presence value (**optional**) neither forces nor forbids the inclusion of the port in the port map. Presence does not apply to interfaces of type **monitor**. See also: 5.10.2.	No	No
SCR 6.9 *OneWireDriver*	Only one component port in a port connection equivalence class may have the direction **out**. See also: 7.3.1.2 and 7.5.3.	No	Yes
SCR 6.10 *OneTransactionalDriver*	Only one component port in a port connection equivalence class may have the initiative **requires**. See also: 7.3.1.2 and 7.5.4.	No	Yes
SCR 6.11 *ExtendedLogPortsExist*	If abstraction definition A extends abstraction definition B, then abstraction definition A needs to have port elements for every port declared in abstraction definition B. If a port in abstraction definition B is not used in bus interfaces using abstraction definition A, then, in abstraction definition A, that port shall have a presence value of **illegal** for all bus interface modes. See also: 5.3.2 and Table 2.	No	Yes
SCR 6.12 *LogicalWireToPhysicalWire*	If the abstraction definition referenced by a bus or abstraction interface specifies a port is a wire port (i.e., the **port** element contains a **wire** subelement), the port map shall only map that logical port to a wire component port. See also: 5.5.2, 6.5.7.2, 6.12.8.2, and 8.6.2.	No	No

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table B.6—Ports *(continued)*

Name	Rule	Single doc check	Post config
SCR 6.13 *LogicalTransToPhysicalTrans*	If the abstraction definition referenced by a bus or abstraction interface specifies a port is a transactional port (i.e., the **port** element contains a **transactional** subelement), the port map shall only map that logical port to a transactional component port. See also: 5.9.2, 6.5.7.2, and 6.12.17.2.	No	No
SCR 6.14 *LogPortSystemRefExists*	The value of the **group** subelement of an **onSystem** element shall match the value of one of the system group names referenced in the bus definition referenced by the abstraction definition containing the **onSystem** element. See also: 5.5.2 and 5.9.2.	No	No
SCR 6.15 *SystemGroupDefined*	The value of the **group** subelement of a **system** element shall match the value of one of the system group names referenced in the bus definition referenced by the bus interface containing the **onSystem** element. See also: 6.5.2.2.	No	No
SCR 6.16 *IsAddressPortNeeded*	If an abstraction definition's **busType** element references an addressable bus, the abstraction definition shall contain at least one **port** element with an **isAddress** subelement. See also: 5.2.2, 5.6.2, and 5.9.2.	No	No
SCR 6.17 *CantMapIllegalPresencePort*	If the abstraction definition defines ports with a presence value of **illegal** for a given interface mode, then the indicated ports may not appear in the port map of a bus interface of that mode type. For system interfaces, the port presence shall be looked up from the **onSystem** element with the group name matching that of the bus or abstraction interfaces. Mirrored bus and abstraction interfaces shall be looked up as if they were not mirrored. Port maps are optional, even on buses with required ports. See also SCR 6.8. The third possible presence value (**optional**) neither forces nor forbids the inclusion of the port in the port map. See also: 5.7.2 and 5.10.2.	No	Yes
SCR 6.18 *PhysicalPortRangeExists*	The range of a **physicalPort** shall be a subset of the range of the referenced port in the component's model element. See also: 6.5.7.2 and B.1.6.	Yes	No
SCR 6.19 *LogicalRangeMatchesPhysical*	Within any **portMap**, the sizes of the ranges of the **physicalPort** and the **logicalPort** shall be equal. See also: 6.5.7.2.	Yes	No
SCR 6.20 *LogicalRangeWithinDefinition*	If the abstraction definition port referenced by a **logicalPort** has a width defined, all elements in the range of the logical port shall be between `width-1` and `0`. See also: 6.5.7.2.	No	No
SCR 6.21 *LogicalBitsMappedOnlyOnce*	Within a single bus interface no logical bit may be mapped more than once, i.e., if two or more **logicalPort** elements for that bus interface reference the same abstraction definition port, their ranges shall not overlap. See also: 6.5.7.2.	Yes	Yes

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table B.6—Ports *(continued)*

Name	Rule	Single doc check	Post config
SCR 6.22 *TransactionalPortBusConnection*	If a transactional port in a component is mapped in a bus interface to a transactional port in an abstraction definition, then the **initiative**, **kind**, **busWidth** and **protocolType** elements in that component port shall match those defined in the abstraction definition's port. See also: 6.5.7.2 and 6.12.17.2.	No	No
SCR 6.23 *TransactionalPortConnection*	Transactional ports shall only be connected together (by an ad hoc connection or through an interconnection) if they have compatible **initiative**, **busWidth**, **kind**, and **protocol** elements, and if none of them contains a **transTypeDefs** element or all of them contain compatible **transTypeDefs** elements. The following definitions apply here. a) Two initiatives with any provides-requires combination are compatible. b) Two kinds are compatible if they are either equal or both are sockets (`simple_socket`, `multi_socket`, or `tlm_socket`). c) Two **busWidth**s are compatible if they are equal on both side of the connection. d) Two protocols are compatible if their **protocolType** are both **tlm**. e) Two protocols are compatible if their **protocolType** are both **custom** and (if defined) their **payload** are the same. f) Two **transTypeDefs** are compatible if their **typeParameters** values are equal and their **serviceTypeDefs typeName** (if defined and not implicit) are the same. See also: 6.5.7.2 and 6.12.17.2.	No	No
SCR 6.24 *CantDriveOutputPort*	A wire port with a direction of **out** shall not have a **driver** element. See also: 6.12.10.2.	Yes	Yes
SCR 6.25 *AdHocPortWidthsMatch*	All ports referenced in an ad hoc connection shall reference the same number of bits. If no **range** is specified for a non-scalar port, then the full range from the port definition is presumed. See also: 7.5.3.	No	No
SCR 6.26 *TiedValueDefaultHasDefault*	All ports references in an ad hoc connection which has a **tiedValue** of **default** shall have a default value defined. See also: 7.5.2.	No	No
SCR 6.27 *ViewlessComponentRestriction*	A **component** without **views** shall contain only phantom ports. See also: 6.12.1.2.	Yes	Yes
SCR 6.28 *VirtualComponentRestriction*	If **isVirtual** is true in a **componentInstantiation** element, then **views** referencing that **componentInstantiation** shall contain only phantom ports. See also: 6.12.3.2.	Yes	Yes

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

B.2.7 Registers

Table B.7—Registers

Name	Rule	Single doc check	Post config
SCR 7.1 *RegisterOverlap*	No register shall have an **addressOffset** that falls within the address range of another register in the same address block, unless one of the registers and their **alternateRegisters** have non-conflicting **access** elements. Non-conflicting **access** elements have a value of **read-only**, **write-only**, or **writeOnce**. The address range of a register is the range `[addressOffset, addressOffset+ ((size + addressBitUnits–1) ÷ addressBitUnits-1)*dim[n-1...0]`, where `dim` is the maximum number of elements for each of `n` dimensions. I.e., registers shall not overlap, unless one is only visible when reading and the other is only visible when writing. See also: 6.11.2.2.	Yes	Yes
SCR 7.2 *BitOverlap*	No bit field shall have a **bitOffset** value that falls within the bit range of another bit field, unless one of the bit fields has an **access** element with the value **read-only** and the other has an **access** element with the value **write-only** or **writeOnce**. The range of a bit field is the range `[bitOffset, bitOffset + width-1]`. I.e., bit fields shall not overlap, unless one is only visible when reading and the other is only visible when writing. See also: 6.11.2.2 and 6.11.8.2.	Yes	Yes
SCR 7.3 *RegisterWithinBlock*	Any register in an address block shall fall entirely within that address block, i.e., for every `register 0 <= addressOffset < addressBlockRange - registerSize`; where **addressBlockRange** is the range of the address block and **registerSize** is the size of the register in LAUs. This is equal to `(((size + addressBitUnits-1) ÷ addressBitUnits))* dim[n-1...0]`, where `dim` is the maximum number of elements for each of `n` dimensions. See also: 6.11.2.2.	Yes	No
SCR 7.4 *BitWithinRegister*	Any bit field in a register shall fall entirely within that register, i.e., for every bit field `0 <= bitOffset <= RegisterSize – bitFieldWidth`; where **RegisterSize** is the **size** (in bits) of the register, and **bitFieldWidth** is the **width** of bit field. See also: 6.11.2.2 and 6.11.8.2.	Yes	Yes
SCR 7.5 *RegisterSizeWithinBlock*	The size of any register shall be no greater than the width of the containing address block. See also: 6.9.6.2.	Yes	Yes

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table B.7—Registers *(continued)*

Name	Rule	Single doc check	Post config
SCR 7.6 *RegisterWithinRegisterFile*	Any register in a register file shall fall entirely within that register file, i.e., for every `register 0 <= register.addressOffset < registerFileRange - registerSize`, where **registerFileRange** is the range of the register file and **registerSize** is the size of the register in LAUs. This is equal to `(((size + addressBitUnits-1) ÷ addressBitUnits))* dim[n-1...0]`, where `dim` is the maximum number of elements for each of `n` dimensions. See also: 6.11.2, 6.11.3, and 6.11.6.	Yes	Yes
SCR 7.7 *RegisterFileWithinBlock*	Any register file in an address block shall fall entirely within that address block, i.e., for every `register file 0 <= registerFile.addressOffset < addressBlockRange - registerFileSize`, where **registerBlockRange** is the range of the address block and **registerFileSize** is the size of the register in LAUs. This is equal to `registerFile.range * dim[n-1...0]`, where `dim` is the maximum number of elements for each of `n` dimensions. See also: 6.9.2.	Yes	Yes
SCR 7.8 *BlockVolatileCondition*	**volatile** cannot be set to **false** for an **addressBlock** where any containing **register** or **field** already has **volatile** set to **true**. See also:6.11.2, 6.11.3, 6.11.8, 6.11.9, and 6.9.3.	Yes	Yes
SCR 7.9 *RegisterVolatileCondition*	**volatile** can not be set to **false** for a **register** where any containing **field** already has **volatile** set to **true**. See also:6.11.2, 6.11.3, 6.11.8, and 6.11.9.	Yes	Yes
SCR 7.10 *FieldUseEnumCondition*	When a **field** has **writeValueConstraint**/**useEnumeratedValues** set to **true**, it also needs to have at least one **enumeratedValue** with the attribute **usage** set to **write** or **read-write**. See also: 6.11.8, 6.11.9, and 6.11.10.	Yes	Yes
SCR 7.11 *FieldConstraintRangeCondition*	When a **field** has a **writeValueConstraint**/**minimum** value and has a **writeValueConstraint**/**maximum** value, the value of **maximum** shall be greater than or equal to the value of **minimum**. See also: 6.11.8, 6.11.9, and 6.11.10.	Yes	Yes
SCR 7.12 *FieldTypeIdentifierCondition*	When multiple **field** elements have the same **typeIdentifier**, the field object shall contain the same contents for the elements in the **fieldDefinitionGroup**. See also: 6.11.8 and 6.11.9.	Yes	Yes
SCR 7.13 *RegisterTypeIdentiferCondition*	When multiple **register** or **alternateRegister** elements have the same **typeIdentifier**, the register object shall contain the same contents for the elements in the **registerDefinitionGroup** or **alternateRegisterDefinitionGroup**. See also: 6.11.3 and 6.11.5.	Yes	Yes

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table B.7—Registers *(continued)*

Name	Rule	Single doc check	Post config
SCR 7.14 *RegisterFileTypeIdentifierCondition*	When multiple **registerFile** elements have the same **typeIdentifier**, the register file object shall contain the same contents for the elements in the **registerFileDefinitionGroup**. See also: 6.11.6.	Yes	Yes
SCR 7.15 *BlockTypeIdentifierCondition*	When multiple **addressBlock** elements have the same **typeIdentifier**, the address block object shall contain the same contents for the elements in the **addressBlockDefinitionGroup**. See also: 6.9.3.	Yes	Yes
SCR 7.16 *noWritePropsInROField*	A register field whose access type does not allow writing (access type **readOnly**) shall not include any of the following sub-elements: **modifiedWriteValue**. See also: 6.11.9.2.	Yes	Yes
SCR 7.17 *noReadPropsInWOField*	A register field whose access type does not allow reading (access types of **writeOnly** or **writeOnce**) shall not include any of the following sub-elements: **readAction**. See also: 6.11.9.2.	Yes	Yes
SCR 7.18 *fieldUsageMatchesAccess*	If the **usage** attribute is defined on an **enumeratedValue** element within a register field, that usage shall be consistent with the **access** type of the register field. See also: 6.11.10.	Yes	Yes
SCR 7.19 *registerFileOverlap*	No register or register file shall have an **addressOffset** that falls within the address range of another register file in the same address block. See also: 6.11.10.	Yes	Yes
SCR 7.20 *resetTypeHARD*	A **resetType** with a name of HARD shall not be specified. See also: 6.19.	Yes	No
SCR 7.21 *resetTypeRefUnspecified*	Only one **resetValue** can be specified without a **resetTypeRef** (indicating the default/**HARD** reset-type) on a **field**. See also: 6.11.8.2.	Yes	No

B.2.8 Memory maps

Table B.8—Memory maps

Name	Rule	Single doc check	Post config
SCR 8.1 *BlockWidthCondition*	The width of an address block included in a memory map shall be a multiple of the memory map's **addressUnitBits**. See also: 6.9.2.2.	Yes	No
SCR 8.2 *NoSubspaceInParallelBank*	Neither a parallel bank, nor banks within a parallel bank, shall contain subspace maps. See also: 6.9.5.2, 6.9.7.2, and 6.9.8.2.	Yes	No

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table B.8—Memory maps *(continued)*

Name	Rule	Single doc check	Post config
SCR 8.3 *ReadOnlyBankCondition*	A **read-only bank** shall only contain **read-only address-Block**s or **bank**s. See also: 6.9.4.2 and 6.11.2.2.	Yes	No
SCR 8.4 *ReadOnlyBlockCondition*	A **read-only addressBlock** shall only contain **read-only register**s. See also: 6.9.4.2 and 6.11.2.2.	Yes	No
SCR 8.5 *ReadOnlyRegisterCondition*	A **read-only register** shall only contain **read-only field**s. See also: 6.11.2.2.	Yes	No
SCR 8.6 *WriteOnlyBankCondition*	A **write-only bank** shall only contain **write-only** or **writeOnc**e **addressBlock**s or **bank**s. See also: 6.9.4.2 and 6.11.2.2.	Yes	No
SCR 8.7 *WriteOnlyBlockCondition*	A **write-only addressBlock** shall only contain **write-only** or **writeOnce register**s. See also: 6.9.4.2 and 6.11.2.2.	Yes	No
SCR 8.8 *WriteOnlyRegisterCondition*	A **write-only register** shall only contain **write-only** or **writeOnce field**s. See also: 6.11.2.2.	Yes	No
SCR 8.9 *addressBlockContent*	A register or register file can appear in an **addressBlock** with a usage of **memory** or **register**. If the **addressBlock** usage is `memory`, then the child registers and register files are interpreted as virtual. See also: 6.9.4.2 and 6.11.2.2.	Yes	No
SCR 8.10 *RW1BankCondition*	A **read-writeOnce bank** shall only contain **read-only**, **read-writeOnce**, or **writeOnce addressBlock**s or **bank**s. See also: 6.9.4.2 and 6.11.2.2.	Yes	No
SCR 8.11 *RW1BlockCondition*	A **read-writeOnce addressBlock** shall only contain **read-only**, **read-writeOnce**, or **writeOnce register**s. See also: 6.9.4.2 and 6.11.2.2.	Yes	No
SCR 8.12 *RW1RegisterCondition*	A **read-writeOnce register** shall only contain **read-only**, **read-writeOnce**, or **writeOnce field**s. See also: 6.11.2.2.	Yes	No
SCR 8.13 *W1BankCondition*	A **writeOnce bank** shall only contain **writeOnce address-Block**s or **bank**s. See also: 6.9.4.2 and 6.11.2.2.	Yes	No
SCR 8.14 *W1BlockCondition*	A **writeOnce addressBlock** shall only contain **writeOnce register**s. See also: 6.9.4.2 and 6.11.2.2.	Yes	No
SCR 8.15 *W1RegisterCondition*	A **writeOnce register** shall only contain **writeOnce field**s. See also: 6.11.2.2.	Yes	No
SCR 8.16 *BlockOverlap*	Two **addressBlock**s in the same **memoryMap** shall not overlap. See also: 6.9.2.2.	Yes	Yes
SCR 8.17 *virtualRegisterContent*	The ***registerDefinitionGroup*** in a virtual register shall only contain **typeIdentifier**, **size**, and **field** elements. See also: 6.9.4.2.	Yes	No

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table B.8—Memory maps *(continued)*

Name	Rule	Single doc check	Post config
SCR 8.18 *virtualAlternateRegisterContent*	When an alternate register is a child of a virtual register, the ***alternateRegisterDefinitionGroup*** shall only contain **typeIdentifier** and **field** elements. See also: 6.11.4.	Yes	No
SCR 8.19 *virtualFieldContent*	If a **field** is a child of a virtual register, the ***fieldData*** group shall only contain **enumeratedValue** and **writeValueConstraint** elements. See also: 6.11.8.	Yes	No

B.2.9 Addressing

Table B.9—Addressing

Name	Rule	Single doc check	Post config
SCR 9.1 *AddressableMasterHasRef*	A non-hierarchical addressable master bus interface shall have an **addressSpaceRef** subelement. See also: 6.5.3.2.	No	No
SCR 9.2 *AddressableSlaveHasMapOrBridge*	A non-hierarchical addressable slave bus interface shall have a **memoryMapRef** subelement or one or more **bridge** subelements referencing addressable master bus interfaces. See also: 6.5.4.2.	No	No
SCR 9.3 *ExecImageCondition*	Only an address space referenced by the **addressSpaceRef** subelement of a **cpu** element may contain an **exectutableImage** subelement. See also: 6.8.1.2 and 6.8.3.2.	No	No
SCR 9.4 *BitSteeringRestriction*	**bitSteering** is not allowed in mirrored-masters, system, or mirrored-system interface modes. See also: 6.5.1.2.	Yes	No
SCR 9.5 *ChannelDataWidthRestriction*	Data widths in a channel shall all be a power 2 multiple of their **bitsInLau**. See also: 6.5.1.2.	Yes	No
SCR 9.6 *BitsInLauRestriction*	**bitsInLau** in a channel shall all be a power 2 multiple of the smallest **bitsInLau**. See also: 6.5.1.2.	Yes	No
SCR 9.7 *SegmentCondition*	For each **segment** within an **addressSpace**, everything between **offsetAddress** and **offsetAddress** + **range**-1 shall be contained within the range of that **addressSpace**. See also: 6.8.1.2 and 6.8.2.2.	Yes	No
SCR 9.8 *SegmentRefExists*	The **segmentRef** needs to reference an existing **segment** of the **addressSpace** in the master referenced by the **masterRef**. See also: 6.9.9.2.	Yes	No

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table B.9—Addressing *(continued)*

Name	Rule	Single doc check	Post config
SCR 9.9 *indirectDataField*	The **field** referenced as **indirectData** shall not be part of the memory-map referenced by the **memoryMapRef**. See also: 6.6.2.	Yes	No
SCR 9.10 *indirectAddressRefField*	The **field** referenced as **indirectAddress** shall not be part of the memory-map referenced by the **memoryMapRef**. See also: 6.6.2.	Yes	No
SCR 9.11 *indirectAddressRefFieldAccess*	The **field** referenced as **indirectAddress** shall not have an access-type of **read-only**, read-**writeOnce**, and **writeOnce**. See also: 6.6.2.	Yes	No

B.2.10 Hierarchy

Table B.10—Hierarchy

Name	Rule	Single doc check	Post config
SCR 10.1 *HierFamilyBusIntfBusTypesMatch*	All members of a hierarchical family of bus interfaces shall reference the same **busDefinition** in their **busType** subelements. They need not reference the same abstraction definitions in their **abstractionType** elements. See also: 7.6.2.	No	No
SCR 10.2 *HierFamilyBusIntfModesMatch*	All members of a hierarchical family of bus interfaces shall have the same interface mode (**master**, **slave**, **system**, etc.). See also: 7.6.2.	No	No
SCR 10.3 *HierFamilyBusIntfConnReqsMatch*	If any member of a hierarchical family of bus interfaces has a **connectionRequired** element with a value of **true**, they all shall have this value. See also: 7.6.2.	No	No
SCR 10.4 *HierFamilyBusIntfSteeringMatch*	If any member of a hierarchical family of bus interfaces has a **bitSteering** element with a value of **on**, they all shall have this value. See also: 7.6.2.	No	No
SCR 10.5 *HierFamilyBusIntfPortMapCondition*	If any member of a hierarchical family of bus interfaces has a **portMap** subelement, they all shall. See also: 7.6.2.	No	No

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

B.2.11 Hierarchy and memory maps

In a hierarchical family of bus interfaces, memory maps should be consistent.

Table B.11—Hierarchy and memory maps

Name	Rule	Single doc check	Post config
SCR 11.1 *hierMapSameAddresses*	In a hierarchical family of slave or mirrored-master bus interfaces, all bus interfaces that define addressing information shall define the same set of addresses to be visible. See also: 7.3.1.	No	No
SCR 11.2 *hierMapSameLocations*	For any member of a hierarchical family of slave or mirrored-master bus interfaces, if an address resolves to reference a location outside the containing hierarchical family of components, that address shall reference the same location (i.e., the same address on the same bus) in every member of the hierarchical family that defines addressing information. See also: 7.3.1.	No	No
SCR 11.3 *hierMapSameProperties*	If any bit address (i.e., address plus bit offset) is resolved to a bit within an address block by any member of a hierarchical family of slave bus interfaces, all members of that family with addressing information shall resolve that bit address to a bit with identical behavioral properties. See also: 7.3.1.	No	No

B.2.12 Constraints

Table B.12—Constraints

Name	Rule	Single doc check	Post config
SCR 12.1 *NoOutputDriveConstraint*	A component wire port with direction **out** shall not have a drive constraint. See also: 6.12.14.	Yes	No
SCR 12.2 *NoInputLoadConstraint*	A component wire port with a direction **in** shall not have a load constraint. See also: 6.12.6.2.	Yes	No
SCR 12.3 *NoOutputDriveModeConstraint*	An **onMaster**, **onSlave**, or **onSystem** element of a wire port with direction **out** shall not contain a drive constraint within its **modeConstraint** element. See also: 6.12.14.	Yes	No
SCR 12.4 *NoInputLoadModeConstraint*	An **onMaster**, **onSlave**, or **onSystem** element of a wire port with direction **in** shall not contain a load constraint within its **modeConstraint** element. See also: 6.12.6.2.	Yes	No

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table B.12—Constraints *(continued)*

Name	Rule	Single doc check	Post config
SCR 12.5 *NoOutputLoadMirroredMode-Constraint*	An **onMaster**, **onSlave**, or **onSystem** element of a wire port with direction **out** shall not contain a load constraint within its **mirroredModeConstraint** element. See also: 6.12.6.2.	Yes	No
SCR 12.6 *NoInputDriveMirroredMode-Constraint*	An **onMaster**, **onSlave**, or **onSystem** element of a wire port with direction **in** shall not contain a drive constraint with its **mirroredModeConstraint** element. See also: 6.12.14.	Yes	No
SCR 12.7 *ConstraintClockExists*	The **clockName** in a timing constraint of a component port shall be the **clockName** of a **clockDriver** or **otherClock-Driver** element within the component unless there are no **clockDriver** or **otherClockDriver** elements present. In that case, the **clockName** shall be the name of another component port. See also: 6.12.14.2.	Yes	No
SCR 12.8 *ConstraintClockLogical PortExists*	The **clockName** in a timing constraint of a port within an abstraction definition shall be the name of another port of the abstraction definition; that referenced port shall have an **isClock** subelement. See also: 5.6.2 and 6.12.14.2.	Yes	No
SCR 12.9 *DriverSingleBitCondition*	Only a scalar port (i.e., a single-bit port) may have a **clock-Driver** or a **singleShotDriver** subelement. See also: 6.12.10.2.	Yes	No

B.2.13 Design configurations

Table B.13—Design configurations

Name	Rule	Single doc check	Post config
SCR 13.1 *ViewConfigInstanceExists*	The value of an **instanceName** within a **viewConfiguration** shall match the value of the **instanceName** element of a **componentInstance** of the design document referenced by the design configuration document containing the **viewConfiguration** element. See also: 10.2.2.	No	No
SCR 13.2 *ViewConfigViewExists*	The value of an **viewName** within a **viewConfiguration** shall match the value of the **name** element of a view within the component referenced by the component instance that is itself referenced by the **instanceName** subelement of the **viewConfiguration** element. See also: 10.2.2.	No	No

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table B.13—Design configurations *(continued)*

Name	Rule	Single doc check	Post config
SCR 13.3 *ViewConfigsUnique*	No two **viewConfiguration** elements within a design configuration shall reference the same view. i.e., no two **viewConfiguration** elements may have the same **instanceName**. See also: 10.2.2.	Yes	Yes
SCR 13.4 *AbstractorInstancesUnique*	No two **abstractor** elements within a design configuration shall have the same **instanceName** element values. The abstractor names shall also not overlap with component instance names within the design. See also: 10.3.2.	Yes	No

B.2.14 Expressions

Table B.14—Expressions

Name	Rule	Single doc check	Post config
SCR 14.1 *unsignedLongintExpression*	A value specified as an **unsignedLongintExpression** shall be resolved to an `unsigned longint` as specified by the SystemVerilog specification. See also: C.3.8.	Yes	No
SCR 14.2 *unsignedPositiveLongintExpression*	A value specified as an **unsignedPositiveLongintExpression** shall be resolved to an `unsigned longint` with a value greater than `0` as specified by the SystemVerilog specification. See also: C.3.10.	Yes	No
SCR 14.3 *signedLongintExpression*	A value specified as a **signedLongintExpression** shall be resolved to a `signed longint` as specified by the SystemVerilog specification. See also: C.3.2.	Yes	No
SCR 14.4 *unsignedIntExpression*	A value specified as an **unsignedIntExpression** shall be resolved to an `unsigned int` as specified by the SystemVerilog specification. See also: C.3.7.	Yes	No
SCR 14.5 *unsignedPositiveIntExpression*	A value specified as an **unsignedPositiveIntExpression** shall be resolved to an `unsigned int` with a value greater than `0` as specified by the SystemVerilog specification. See also: C.3.9.	Yes	No
SCR 14.6 *realExpression*	A value specified as a **realExpression** shall resolve to a `real` as specified by the SystemVerilog specification. See also: C.3.1.	Yes	No

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table B.14—Expressions *(continued)*

Name	Rule	Single doc check	Post config
SCR 14.7 *stringExpression*	A value specified as a **stringExpression** shall be resolved to a `string` as specified by the SystemVerilog specification. See also: C.3.3.	Yes	No
SCR 14.8 *stringURIExpression*	A value specified as a **stringURIExpression** shall be resolved to a `string` as specified by the SystemVerilog specification after environment-variables have been substituted. See also: C.3.4.	Yes	No
SCR 14.9 *unsignedBitExpression*	A value specified as an **unsignedBitExpression** shall be resolved to an `unsigned bit` as specified by the SystemVerilog specification. See also: C.3.5.	Yes	No
SCR 14.10 *unsignedBitVectorExpression*	A value specified as an **unsignedBitVectorExpression** shall be resolved to an `unsigned bit vector` as specified by the SystemVerilog specification, where the vector size is determined by an external value (e.g., field-size for reset-value). See also: C.3.6.	Yes	No
SCR 14.11 *parameterExpression*	A parameter expression (***complexBaseExpression***) shall be resolved to the SystemVerilog `type` and `sign` specified for the specific parameter. See also: C.3.	Yes	No

B.2.15 Presence

In Table B.15, an element is considered to be *present* if it contains an **isPresent** immediate child element whose value resolved to `1` during configuration. An element is consider to be *nonpresent* if it contains an **isPresent** immediate child element whose value resolves to `0` during configuration. An element enclosed within a *nonpresent* ancestor element is considered *nonpresent* even if the descendant element has an **isPresent** sub-element that evaluates to `1`.

Table B.15—Presence

Name	Rule	Single doc check	Post config
SCR 15.1 *npSubSPcMasterInt*	A *present* **subspaceMap** may not contain a **masterRef** that references a *nonpresent* busInterface. See also: 6.9.9.	Yes	Yes
SCR 15.2 *npPortMapPhysPort*	A *present* **portMap** within a present **busInterface** may not reference a *nonpresent* model port. See also: 6.5.7.	Yes	Yes
SCR 15.3 *npPortMapLogPort*	A *present* **portMap** within a present **busInterface** may not reference a *nonpresent* logical port See also: 6.5.7.	No	Yes

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table B.15—Presence *(continued)*

Name	Rule	Single doc check	Post config
SCR 15.4 *npBridgeMasterInt*	A *present* **bridge** may not refer to a *nonpresent* master interface. See also: 6.4.2.	Yes	Yes
SCR 15.5 *npChanBusInt*	A *present* **busInterfaceRef** within a present channel may not reference a *nonpresent* **busInterface**. See also: 6.7.2.	Yes	Yes
SCR 15.6 *tooFewChanBusIntRefs*	A *present* **channel** must contain at least two *present* **busInterfaceRef** elements. See also: 6.7.2.	Yes	Yes
SCR 15.7 *npMasterAddrSpace*	A *present* master **busInterface** may not contain an **addressSpaceRef** that references a *nonpresent* **addressSpace**. See also: 6.5.3.2.	Yes	Yes
SCR 15.8 *npSubSpcSeg*	A *present* **subspaceMap** may not contain a **segmentRef** referencing a *nonpresent* **segment**. See also: 6.9.9.2.	Yes	Yes
SCR 15.9 *npSlaveMemMap*	A *present* **slave** interface may not contain a **memoryMapRef** that references a *nonpresent* **memoryMap**. See also: 6.5.4.2.	Yes	Yes
SCR 15.10 *npIntHierCompInst*	A configured design may not contain any *present* interconnection with a **hierInterface** that references a *nonpresent* component bus interface. See also: 7.4.2.	No	Yes
SCR 15.11 *npViewConfCompInst*	A *present* **viewConfiguration**'s **instanceName** element may not reference a *nonpresent* **componentInstance**. See also: 10.5.2.	No	Yes
SCR 15.12 *npViewConfCompView*	A *present* **viewConfiguration**'s **view** element may not reference a *nonpresent* component view. See also: 10.5.2.	No	Yes
SCR 15.13 *npModParm*	A **componentInstance** in a design may not include a **configurableElementValue** whose **referenceId** refers to a *nonpresent* parameter. See also: 7.2.2.	No	Yes
SCR 15.14 *npCpuAddrSpace*	A *present* **addressSpaceRef** may not refer to a *nonpresent* **addressSpace**. See also: 6.5.3.2.	Yes	Yes
SCR 15.15 *npIntCompInst*	A *present* interconnection may not reference a *nonpresent* **componentInstance** via any of its **activeInterface**s. See also: 7.3.1.2.	Yes	Yes
SCR 15.16 *npIntBusInt*	A *present* interconnection may not reference a *nonpresent* **busInterface** via any of its **activeInterface**s. See also: 7.3.1.2.	No	Yes
SCR 15.17 *npIntConfInt*	A *present* **interconnectionRef** may not refer to a *nonpresent* interconnection within the design configured by the **designConfiguration** element containing the **interconnectionRef**. See also: 10.3.2.	No	Yes

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table B.15—Presence *(continued)*

Name	Rule	Single doc check	Post config
SCR 15.18 *npAbstractorIntRef*	A *present* **interfaceRef** may not contain a **componentRef** referring to a *nonpresent* **componentInstance** or **busRef** referring to a *nonpresent* **busInterface**. See also: 10.3.2.	No	Yes
SCR 15.19 *npMonActCompInst*	A *present* **monitorInterconnection** may not reference a *nonpresent* **componentInstance** via its **monitoredActiveInterface** element. See also: 7.3.2.	Yes	Yes
SCR 15.20 *npMonActBusInt*	A *present* **monitorInterconnection** may not reference a *nonpresent* **busInterface** via its **monitoredActiveInterface** element. See also: 7.3.2.	No	Yes
SCR 15.21 *npMonCompInst*	A *present* **monitorInterconnection** may not reference a *nonpresent* **componentInstance** via its **monitorInterface** element. See also: 7.3.2.	Yes	Yes
SCR 15.22 *npMonBusInt*	A *present* **monitorInterconnection** may not reference a *nonpresent* **busInterface** via its **monitorInterface** element. See also: 7.3.2.	No	Yes
SCR 15.23 *npAdHocCompInst*	A *present* **adHocConnection** may not reference a *nonpresent* **componentInstance** via any of its **internalPortRefernce** elements See also: 7.5.2.	Yes	Yes
SCR 15.24 *npAdHocPort*	A *present* **adHocConnection** may not reference a *nonpresent* port via any of its **internalPortRefernce** elements. See also: 7.5.2.	No	Yes
SCR 15.25 *npAdHocHierPort*	A *present* **adHocConnection** may not reference a *nonpresent* port via any of its **externalPortRefernce** elements. See also: 7.5.2.	No	Yes

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

B.2.16 Access handles

Table B.16—Access handles

Name	Rule	Single doc check	Post config
SCR 16.1 *accessHandleIndex*	The indices specified for the **accessHandle** should reference an index within the bounds specified by the **dim** element when specified for a register or register-file and **array** element when specified for a port. See also: C.1.3.2 and C.1.4.2.	Yes	Yes
SCR 16.2 *accessHandleSlice*	It is not allowed to specify more than one **accessHandle**/**slices**/**slice** element and it is not allowed to specify a **slice**/**range** on an **addressBlock** with a usage of **register** or **reserved**. See also: C.1.2.2, C.1.3.2, and C.1.4.2.	Yes	Yes
SCR 16.3 *WhiteBoxElementRefExists*	A **whiteboxElementRef**, which references a **whiteboxElement** with a **whiteboxType** of **pin**, shall have a **pathName** that is a **port** in the containing description. See also: 6.17.	Yes	No

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Annex C

(normative)

Common elements and concepts

This annex details common elements and concepts that appear many times throughout the standard.

C.1 accessHandles

All memory mapped objects can specify a list of **accessHandle** elements. Each **accessHandle** stores a portion of the HDL path for the parent object. There is a list of **accessHandle**s associated with each memory mapped object to

a) store a different HDL path for each view of a component (see 6.12.1.2). This is specified by the **accessHandle viewRef**.

b) indicate there are multiple copies of the memory mapped object within the specified view. This is indicated by two or more **accessHandle** elements which have the same **viewRef** or with no **viewRef** child elements. See C.26.

The HDL path for a particular memory mapped object (e.g., a field) is distributed across the memory map hierarchy. To calculate the HDL path relative to the parent component, it is necessary to traverse from the leaf memory map object up the memory map hierarchy, concatenating the HDL path stored in the **accessHandle** associated with each memory map object.

There are several types of **accessHandle** elements:

— A **simpleAccessHandle** is attached to hierarchical memory mapped objects, i.e., **bank** elements. See C.1.1.

— A **nonIndexedLeafAccessHandle** is attached to memory mapped objects without a dimension, i.e., **addressBlock** and **field** elements. See C.1.2.

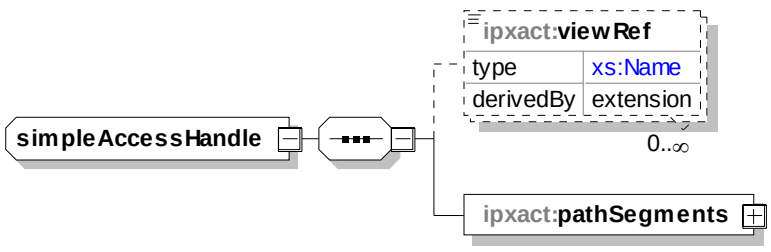
— An **indexedAccessHandle** is attached to memory mapped objects with a dimension, i.e., **alternateRegister**, **register**, and **registerFile** elements. See C.1.3.

— A **leafAccessHandle** describes how to access the associated IP-XACT object. See C.1.4.

C.1.1 simpleAccessHandle

C.1.1.1 Schema

The following schema details the information contained in an **accessHandle** of type ***simpleAccessHandle***.



Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

C.1.1.2 Description

The ***simpleAccessHandle*** stores a portion of a HDL path for a memory mapped object. It contains the following elements.

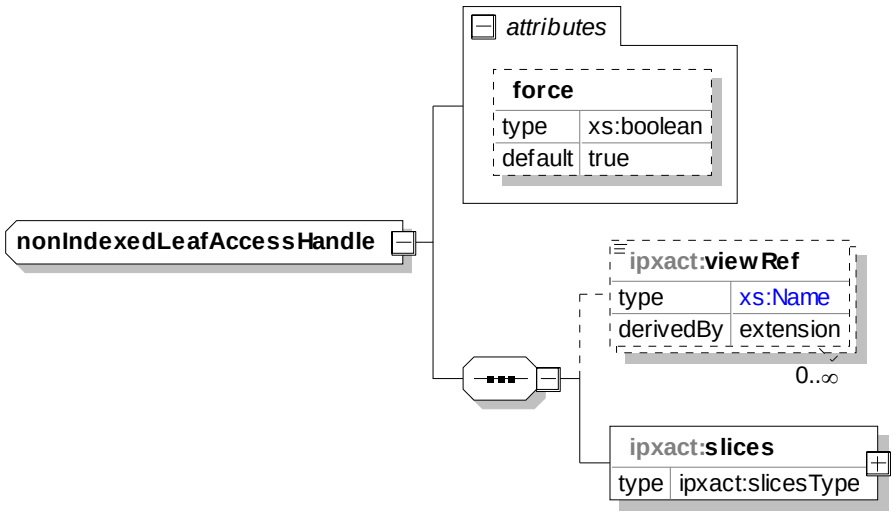
a) **viewRef** (optional; type: ***Name***) specifies the list of one or more views to which this **accessHandle** applies. If none are specified, the **accessHandle** is presumed to apply to all views. See C.26.

b) **pathSegments** (mandatory) is a language-independent mechanism for referencing variables in a view. See C.21.

C.1.2 nonIndexedLeafAccessHandle

C.1.2.1 Schema

The following schema details the information contained in an **accessHandle** of type ***nonIndexedLeafAccessHandle***.



C.1.2.2 Description

A ***nonIndexedLeafAccessHandle*** stores a portion of a HDL path for a memory mapped object and enables slicing in the HDL path. The ***nonIndexedLeafAccessHandle*** type contains the following attributes and elements.

a) **force** (optional; type: ***boolean***; default: **true**) indicates if it is possible to directly write to this object via a back door access.

b) **viewRef** (optional; type: ***Name***) specifies the list of one or more views to which this **accessHandle** applies. If none are specified, the **accessHandle** is presumed to apply to all views. See C.26.

c) **slices** (mandatory) specifies a list of **slice** elements. See C.1.5.
Multiple **slice** elements shall not be used when the parent **accessHandle** is associated with an **addressBlock** with a type of **register** or **reserved**.

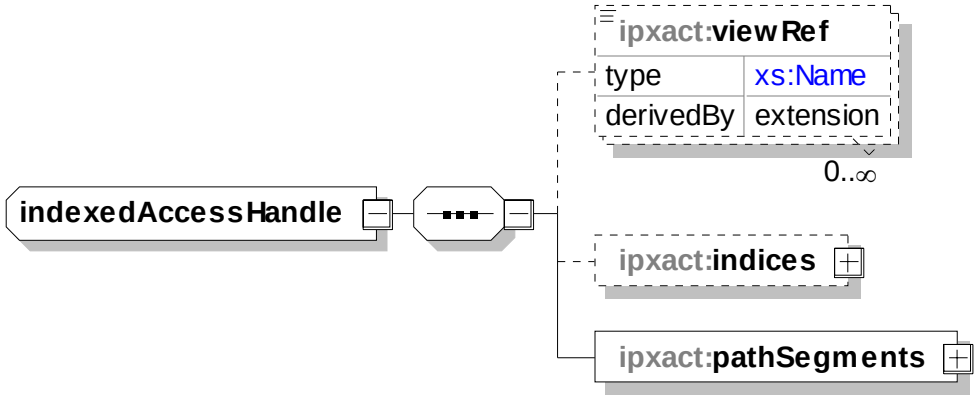
See also: SCR 16.2.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

C.1.3 indexedAccessHandle

C.1.3.1 Schema

The following schema details the information contained in an **accessHandle** of type ***indexedAccessHandle***.



C.1.3.2 Description

An ***indexedAccessHandle*** stores a portion of a HDL path for multi-dimensional memory mapped objects. It enables specifying a HDL path for one type of a memory mapped object. The ***indexedAccessHandle*** element contains the following attributes and elements.

a) **viewRef** (optional; type: ***Name***) specifies the list of one or more views to which this **accessHandle** applies. If none are specified, the **accessHandle** is presumed to apply to all views. See C.26.

b) **indices** (optional) specifies a list of **index** elements. The **indices** specify an element in the IP-XACT object to which this **accessHandle** applies. See C.9.

c) **pathSegments** (mandatory) is a language-independent mechanism for referencing variables in a view. See C.21.

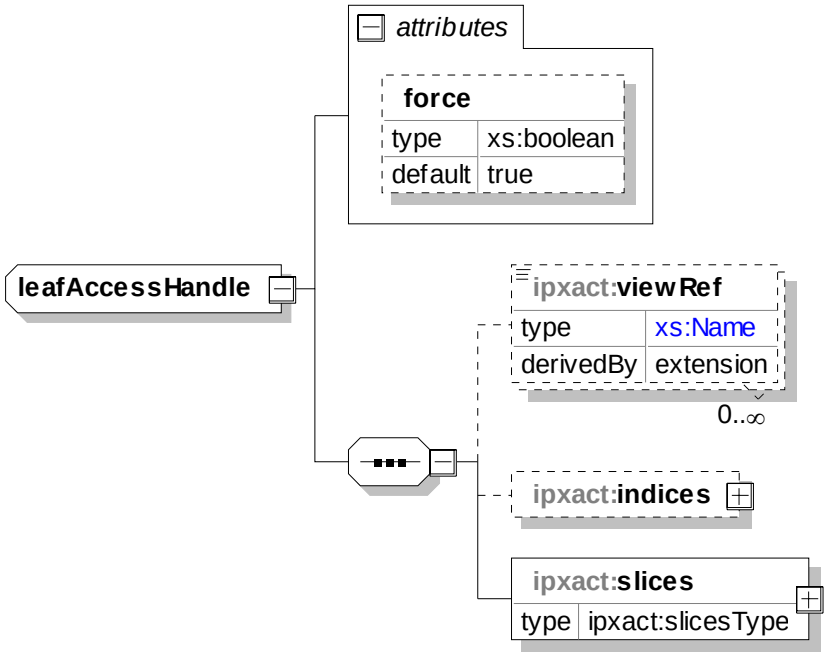
See also: SCR 16.1 and SCR 16.2.

C.1.4 leafAccessHandle

C.1.4.1 Schema

The following schema details the information contained in an **accessHandle** of type ***leafAccessHandle***.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



C.1.4.2 Description

A ***leafAccessHandle*** describes how to access the associated IP-XACT object e.g., a **port** in a **view** or **views**. The ***leafAccessHandle*** type contains the following attributes and elements.

a) **force** (optional; type: ***boolean***; default: **true**) indicates if it is possible to directly write to this object via a back door access.

b) **viewRef** (optional; type: ***Name***) specifies the list of one or more views to which this **accessHandle** applies. If none are specified, the **accessHandle** is presumed to apply to all views. See C.26.

c) **indices** (optional) specifies a list of **index** elements. The **indices** specify an element in the IP-XACT object to which this **accessHandle** applies. See C.9.

d) **slices** (mandatory) specifies a list of slices. See C.1.5.

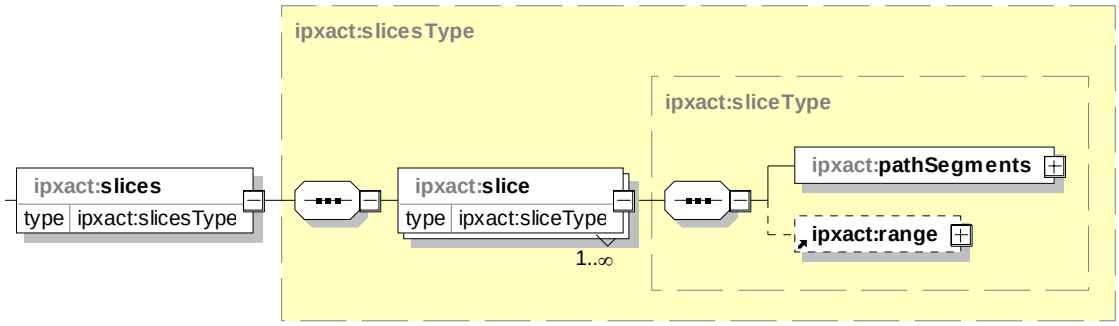
See also: SCR 16.1 and SCR 16.2.

C.1.5 slices

C.1.5.1 Schema

The following schema details the information contained in the **slices** element.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



C.1.5.2 Description

The **slices** element specifies a list of **slice**s. A **slice** consists of one or more **pathSegments**. If only one **slice** is present, the IP-XACT object is represented as one variable in the view. If there are multiple **slice**s, the IP-XACT object is represented by several variables in the view. In this case, the calculated path for each **slice** is concatenated in order to define the total set of bits in the IP-XACT object. Concatenation is from `msb` to `lsb`. The **slice** element contains the following elements.

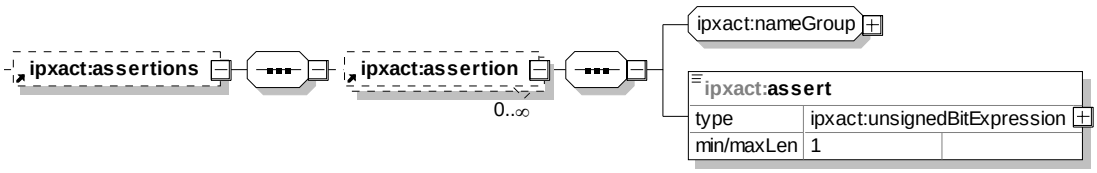
a) **pathSegments** (mandatory) is a language independent mechanism for referencing variables in a view. See C.21.

b) **range** (optional) corresponds to a range (bit select) into a variable in the RTL, i.e., the referenced view. See C.22.

C.2 assertions

C.2.1 Schema

The following schema details the information contained in the **assertions** element.



C.2.2 Description

The **assertions** element contains an unbounded list of **assertion** elements. An **assertion** describes the allowed parameter values as an assertion expression. If this expression evaluates to *False*, the **name**, **displayName**, and/or **description** (of the ***nameGroup***) can be used to relay that the assertion failed. The **assertion** element definition contains the following elements.

a) ***nameGroup*** is defined in C.12.

b) **assert** (mandatory; type: ***unsignedBitExpression*** (see C.3.5)) contains an expression that is expected to evaluate to *True*.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

See also: SCR 5.26.

C.3 complexBaseExpression

The ***complexBaseExpression*** is used as the base type for the more specific expression types, but can also be used directly by parameters. When used directly on a parameter, the value of the element needs to be specified as an expression which can resolve to the type and signed-ness specified for the containing parameter. When the value does not represent the type specified, the parser should attempt to cast the value to this type. An error should be reported when the value cannot be cast.

The **minimum** and **maximum** attributes can be used to further constrain the value; the type and signed-ness of the minimum and maximum value corresponds to the type of the parameter and signed-ness. An error should be reported when the value is specified outside the minimum and maximum range.

complexBaseExpression can be any of the following subelements:

- ***realExpression*** (see C.3.1)
- ***signedLongintExpression*** (see C.3.2)
- ***stringExpression*** (see C.3.3)
- ***stringURIExpression*** (see C.3.4)
- ***unsignedBitExpression*** (see C.3.5)
- ***unsignedBitVectorExpression*** (see C.3.6)
- ***unsignedIntExpression*** (see C.3.7)
- ***unsignedLongintExpression*** (see C.3.8)
- ***unsignedPositiveIntExpression*** (see C.3.9)
- ***unsignedPositiveLongintExpression*** (see C.3.10)

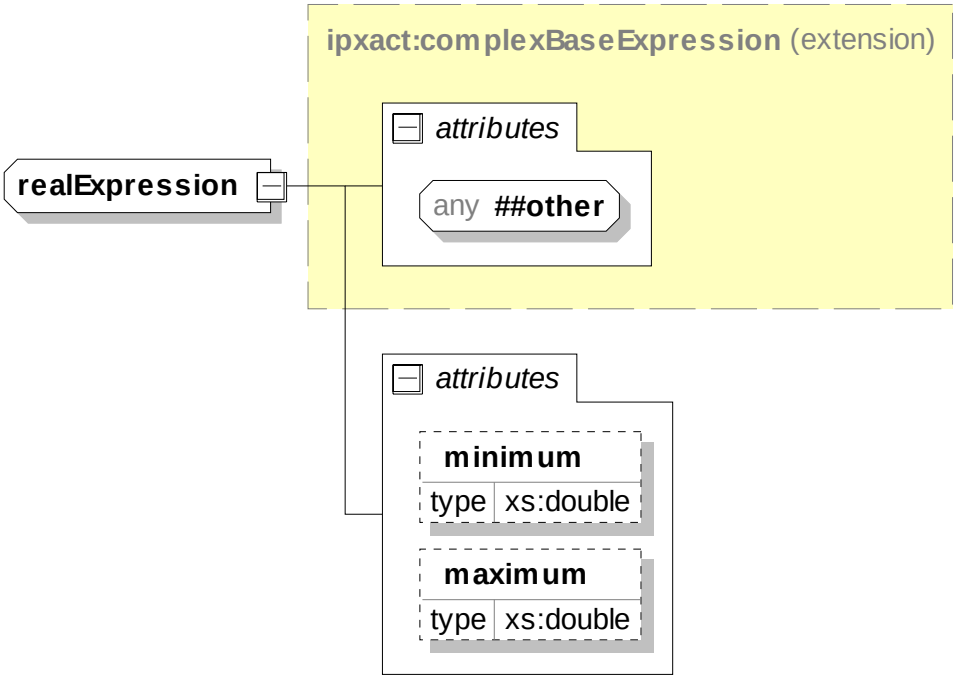
See also: SCR 5.13 and SCR 14.11.

C.3.1 realExpression

C.3.1.1 Schema

The following schema details the information contained in the ***realExpression*** element.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



C.3.1.2 Description

The value of the ***realExpression*** element needs to be specified as an expression which can resolve to a 64-bit `real` value (as specified by IEEE Std 1800). When the value does not represent a `real`, the parser should attempt to cast the value. An error should be reported when the value cannot be cast.

The **minimum** and **maximum** attributes can be used to further constrain the value. An error should be reported when the value is specified outside the minimum and maximum range.

a) **minimum** (optional) indicates the lower bound for the value of the containing element.

b) **maximum** (optional) indicates the upper bound for the value of the containing element.

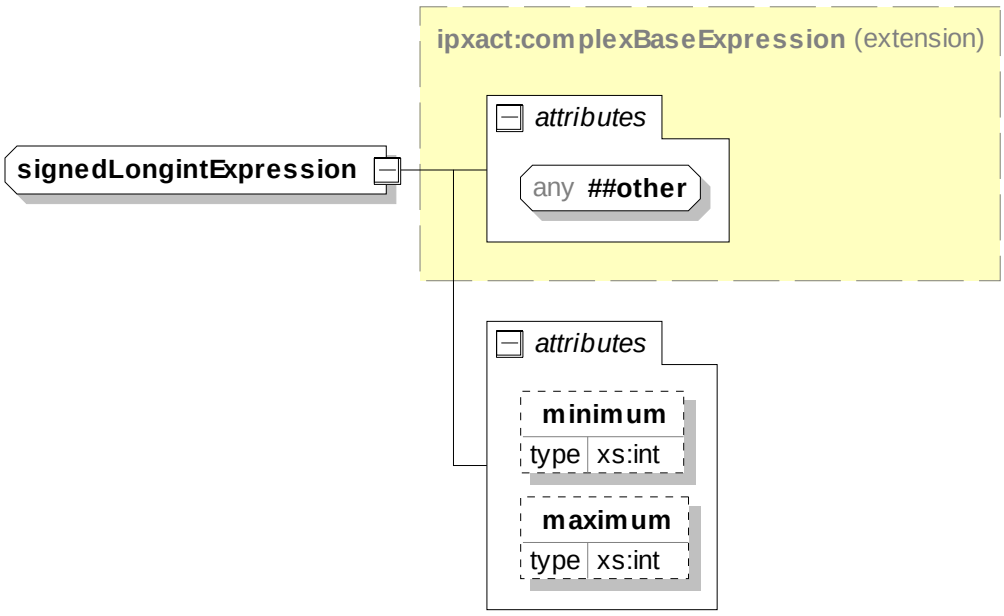
See also: SCR 14.6.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

C.3.2 signedLongintExpression

C.3.2.1 Schema

The following schema details the information contained in the ***signedLongintExpression*** element.



C.3.2.2 Description

The value of the ***signedLongintExpression*** element needs to be specified as an expression which can resolve to a 64-bit `signed longint` value (as specified by IEEE Std 1800). When the value does not represent a `signed longint`, the parser should attempt to cast the value. An error should be reported when the value cannot be cast.

The **minimum** and **maximum** attributes can be used to further constrain the value. An error should be reported when the value is specified outside the minimum and maximum range.

a) **minimum** (optional) indicates the lower bound for the value of the containing element.

b) **maximum** (optional) indicates the upper bound for the value of the containing element.

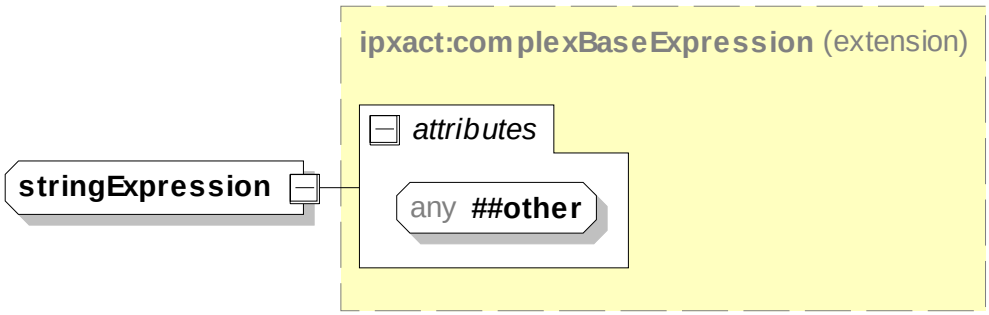
See also: SCR 14.3.

C.3.3 stringExpression

C.3.3.1 Schema

The following schema details the information contained in the ***stringExpression*** element.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



C.3.3.2 Description

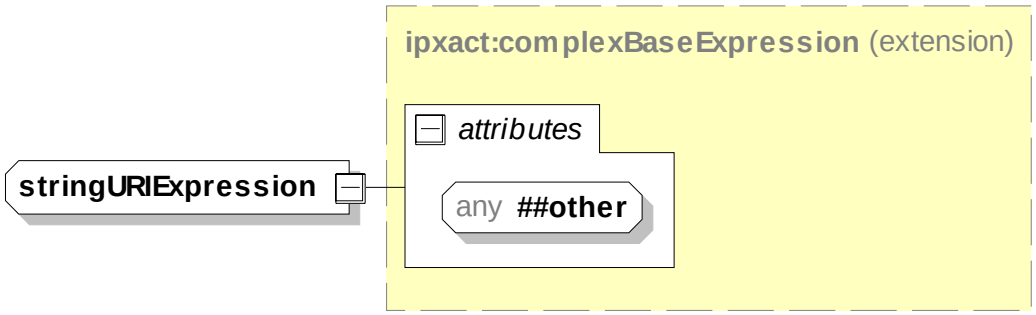
The value of the ***stringExpression*** element needs to be specified as an expression which can resolve to a `string` value (as specified by IEEE Std 1800). An error should be reported when the value does not represent a `string`.

See also: SCR 14.7.

C.3.4 stringURIExpression

C.3.4.1 Schema

The following schema details the information contained in the ***stringURIExpression*** element.



C.3.4.2 Description

The value of the ***stringURIExpression*** element needs to be specified as an expression which can resolve to a `URI` value (as specified by IEEE Std 1800). This value can contain environment variables in the `${}` form which should be resolved before evaluating the URI. An error should be reported when the value does not represent a `URI`.

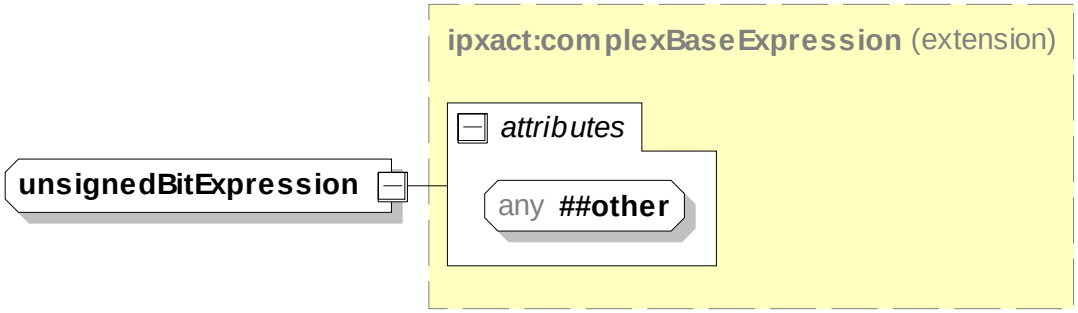
See also: SCR 14.8.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

C.3.5 unsignedBitExpression

C.3.5.1 Schema

The following schema details the information contained in the ***unsignedBitExpression*** element.



C.3.5.2 Description

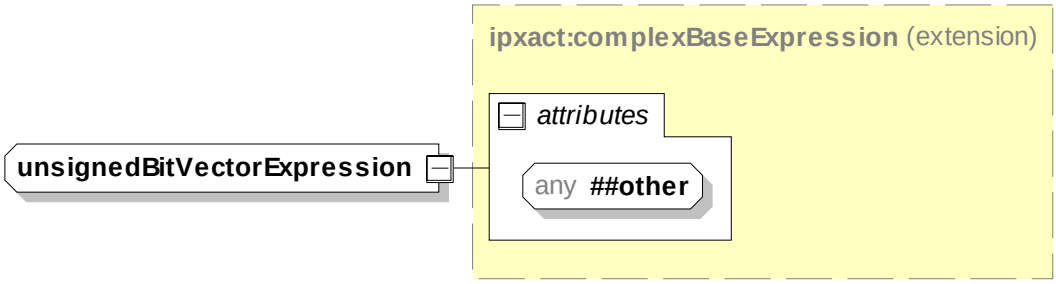
The value of the ***unsignedBitExpression*** element needs to be specified as an expression which can resolve to a 1-bit `bit` value (`1` or `0`) (as specified by IEEE Std 1800). When the value does not represent a `bit`, the parser should attempt to cast the value. An error should be reported when the value cannot be cast.

See also: SCR 14.9.

C.3.6 unsignedBitVectorExpression

C.3.6.1 Schema

The following schema details the information contained in the ***unsignedBitVectorExpression*** element.



C.3.6.2 Description

The value of the ***unsignedBitVectorExpression*** element needs to be specified as an expression which can resolve to a vector of `bits` (as specified by IEEE Std 1800). The length of the `bit vector` is based on the width of the containing object, i.e., the port or field.When the value does not represent a `bit vector`, the parser should attempt to cast the value. An error should be reported when the value cannot be cast.

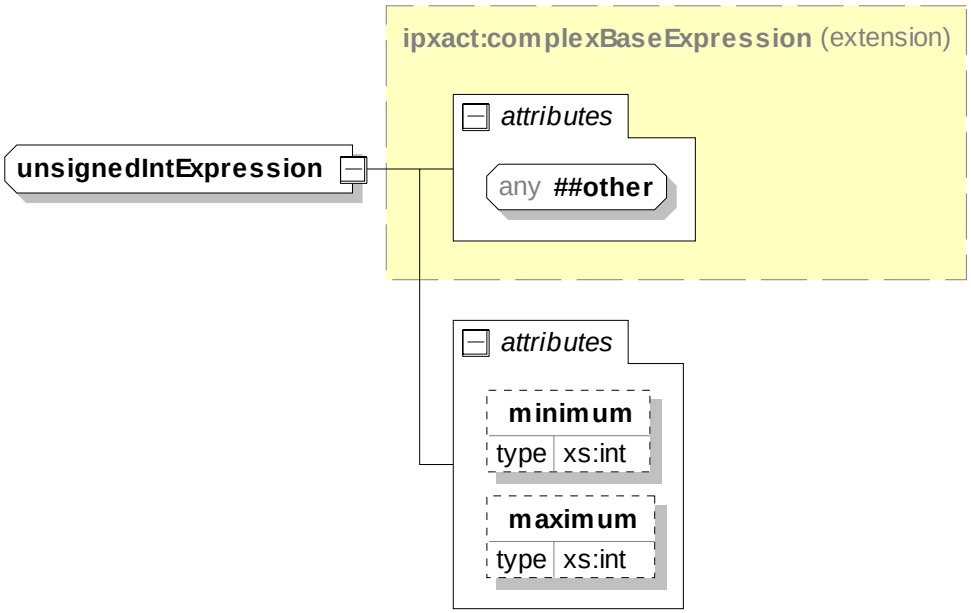
Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

See also: SCR 14.10.

C.3.7 unsignedIntExpression

C.3.7.1 Schema

The following schema details the information contained in the ***unsignedIntExpression*** element.



C.3.7.2 Description

The value of the ***unsignedIntExpression*** element needs to be specified as an expression which can resolve to a ~~64~~32-bit `unsigned int` value (as specified by IEEE Std 1800). When the value does not represent an `unsigned int`, the parser should attempt to cast the value. An error should be reported when the value cannot be cast.

The **minimum** and **maximum** attributes can be used to further constrain the value. An error should be reported when the value is specified outside the minimum and maximum range.

a) **minimum** (optional) indicates the lower bound for the value of the containing element.

b) **maximum** (optional) indicates the upper bound for the value of the containing element.

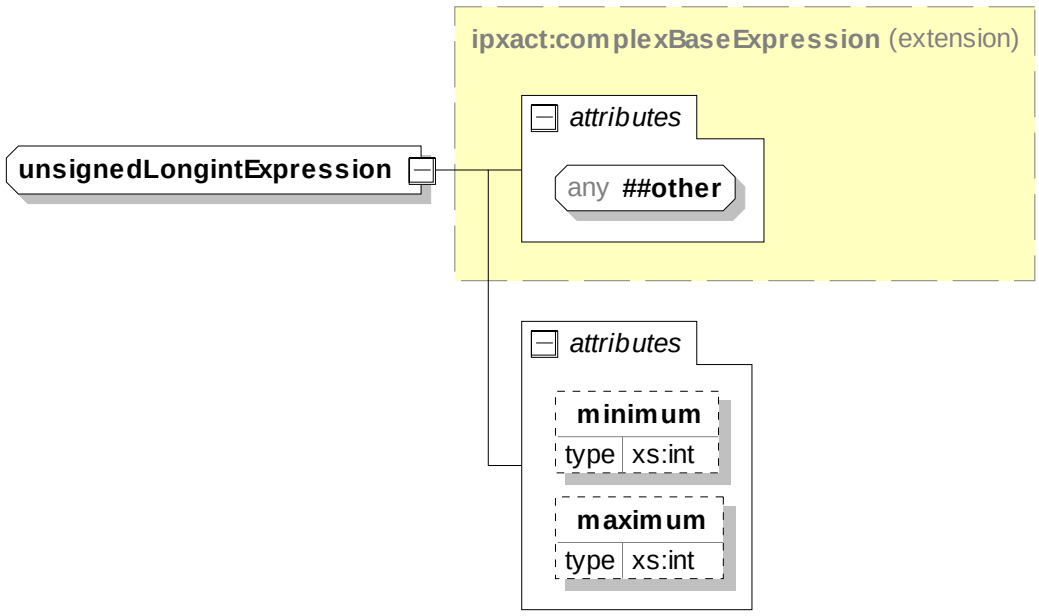
See also: SCR 14.4.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

C.3.8 unsignedLongintExpression

C.3.8.1 Schema

The following schema details the information contained in the ***unsignedLongintExpression*** element.



C.3.8.2 Description

The value of the ***unsignedLongintExpression*** element needs to be specified as an expression which can resolve to a 64-bit `unsigned longint` value (as specified by IEEE Std 1800). When the value does not represent an `unsigned longint`, the parser should attempt to cast the value. An error should be reported when the value cannot be cast.

The **minimum** and **maximum** attributes can be used to further constrain the value. An error should be reported when the value is specified outside the minimum and maximum range.

a) **minimum** (optional) indicates the lower bound for the value of the containing element.

b) **maximum** (optional) indicates the upper bound for the value of the containing element.

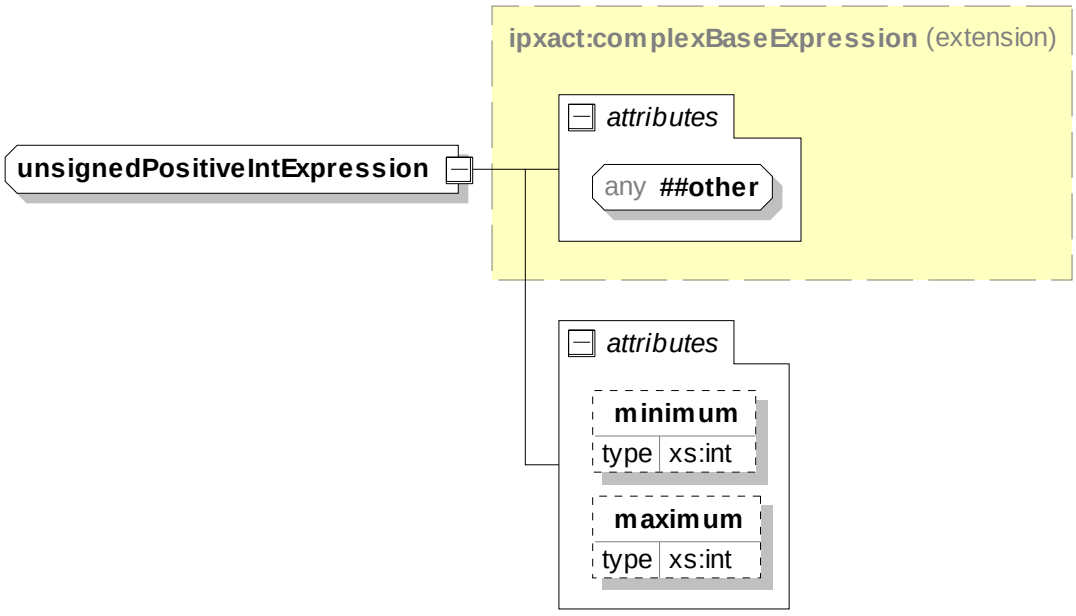
See also: SCR 14.1.

C.3.9 unsignedPositiveIntExpression

C.3.9.1 Schema

The following schema details the information contained in the ***unsignedPositiveIntExpression*** element.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



C.3.9.2 Description

The value of the ***unsignedPositiveIntExpression*** element needs to be specified as an expression which can resolve to a ~~64~~32-bit `unsigned positive int` value (as specified by IEEE Std 1800). When the value does not represent an `unsigned int`, the parser should attempt to cast the value. An error should be reported when the value cannot be cast or it is not a positive number.

The **minimum** and **maximum** attributes can be used to further constrain the value. An error should be reported when the value is specified outside the minimum and maximum range.

a) **minimum** (optional) indicates the lower bound for the value of the containing element.

b) **maximum** (optional) indicates the upper bound for the value of the containing element.

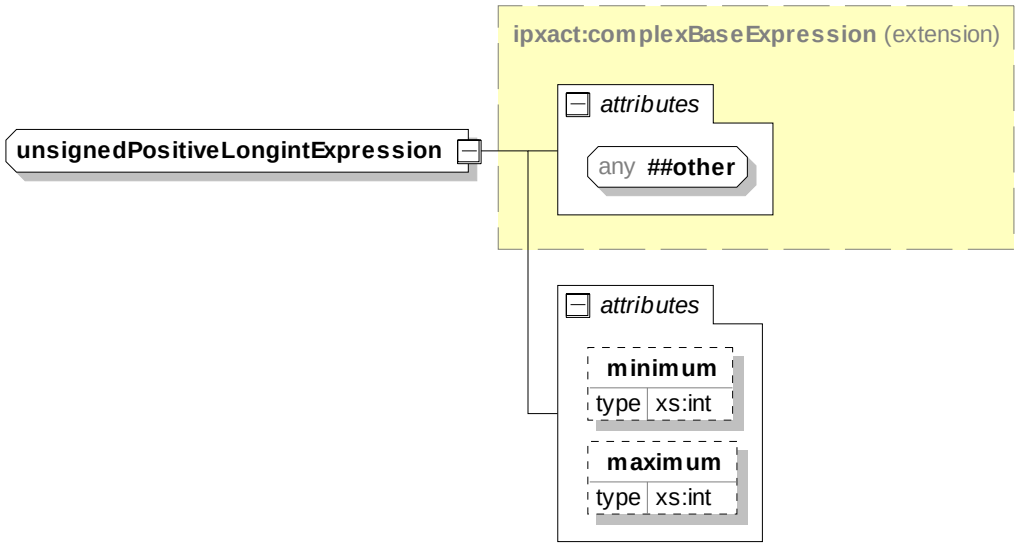
See also: SCR 14.5.

C.3.10 unsignedPositiveLongintExpression

C.3.10.1 Schema

The following schema details the information contained in the ***unsignedPositiveLongintExpression*** element.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



C.3.10.2 Description

The value of the ***unsignedPositiveLongintExpression*** element needs to be specified as an expression which can resolve to a 64-bit `unsigned positive longint` value (as specified by IEEE Std 1800). When the value does not represent an `unsigned longint`, the parser should attempt to cast the value. An error should be reported when the value cannot be cast or it is not a positive number.

The **minimum** and **maximum** attributes can be used to further constrain the value. An error should be reported when the value is specified outside the minimum and maximum range.

a) **minimum** (optional) indicates the lower bound for the value of the containing element.

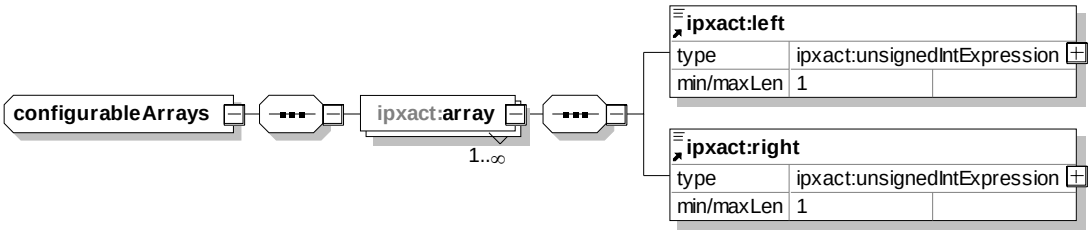
b) **maximum** (optional) indicates the upper bound for the value of the containing element.

See also: SCR 14.2.

C.4 configurableArrays

C.4.1 Schema

The following schema details the information contained in the **configurableArrays** type.



Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

C.4.2 Description

The **configurableArrays** type specifies a set of **array**s. An **array** element contains the following elements.

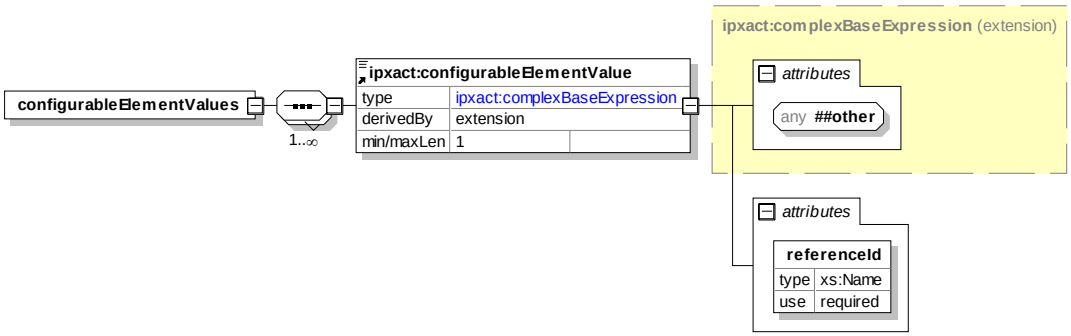
a) **left** (mandatory; type: ***unsignedIntExpression*** (see C.3.8)) specifies the left range for the bit slice used to map a port vector to the bus interface.

b) **right** (mandatory; type: ***unsignedIntExpression*** (see C.3.8)) specifies the right range for the bit slice used to map a port vector to the bus interface.

C.5 configurableElementValues

C.5.1 Schema

The following schema details the information contained in the ***configurableElementValues*** element.



C.5.2 Description

The **configurableElementValues** element defines the configuration for a specific component instance by providing the value of a specific component parameter. The **configurableElementValues** is an unbounded list of **configurableElementValue** elements.

a) **configurableElementValue** (mandatory; type: ***complexBaseExtension*** (see C.3)) is an unbounded list that specifies the value to apply to a configurable element; in this instance, it is pointed to by the **referenceId** attribute.

b) The contained **referenceId** (mandatory; type: ***Name***) attribute is a reference to the **parameterId** attribute of an element in the component instance.

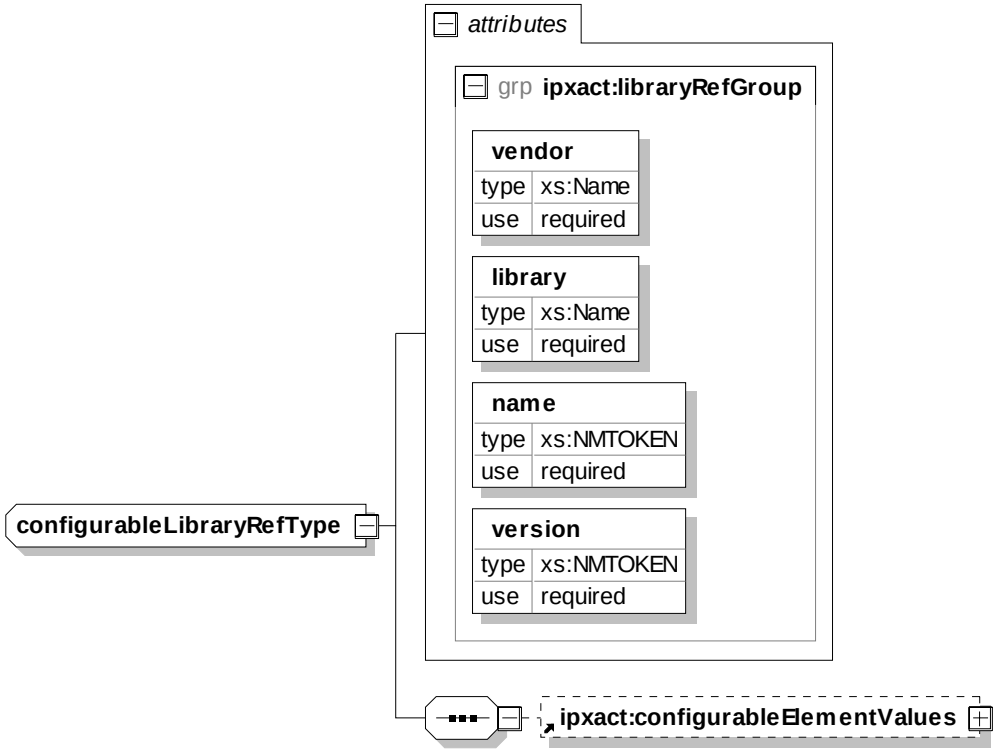
See also: SCR 5.4, SCR 5.5, SCR 5.6, SCR 5.7, SCR 5.8, SCR 5.9, SCR 5.10, and SCR 5.11.

C.6 configurableLibraryRefType

C.6.1 Schema

The following schema details the information contained in the ***configurableLibraryRefType*** type.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



C.6.2 Description

The ***configurableLibraryRefType*** type defines a set of four attributes that reference another IP-XACT description through the unique VLNV identifier and any configuration information.

a) The **vendor** attribute (mandatory; type: ***Name***) identifies the owner of the referenced description.

b) The **library** attribute (mandatory; type: ***Name***) identifies the library of the referenced description.

c) The **name** attribute (mandatory; type: ***NMTOKEN***) identifies the name of the referenced description.

d) The **version** attribute (mandatory; type: ***NMTOKEN***) identifies the version of the referenced description.

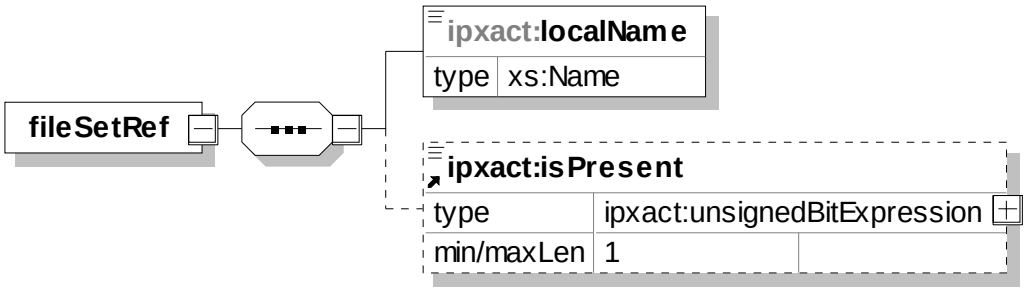
e) **configurableElementValues** (optional) specifies the configuration for a specific component instance by providing the value of a specific component parameter. See C.5.

C.7 fileSetRef

C.7.1 Schema

The following schema details the information contained in the **fileSetRef** element.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



C.7.2 Description

The **fileSetRef** element defines a reference to a **fileSet** contained in the containing document. The **fileSetRef** element contains the following element.

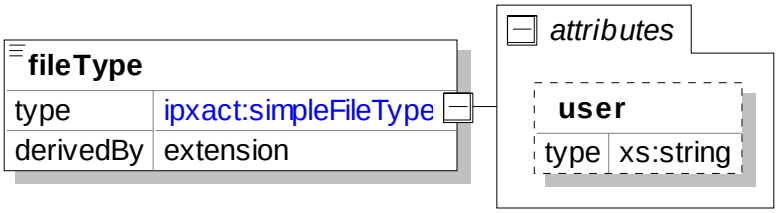
a) **localName** (mandatory; type: ***Name***) shall contain a name of a **fileSet**/**name** within the local description.

b) The **isPresent** (optional; type: ***unsignedBitExpression*** (see C.3.5); default: **1**) element defines whether the enclosing element is present in the document. See C.10.

C.8 fileType

C.8.1 Schema

The following schema details the information contained in the ***fileType*** element.



C.8.2 Description

The ***fileType*** element defines the format of a referenced file. The ***fileType*** group contains one or more of the following elements and attributes.

a) **fileType** (mandatory) describes the type of file referenced from this enumerated list of industry standard files types.

1) **unknown**

2) **asmSource**, **cSource**, **cppSource**, **eSource**, **OVASource**, **perlSource**, **pslSource**, **SVASource**, **tclSource**, **veraSource**, **systemCSource**, **systemCSource-2.0**, **systemCSource-2.0.1**, **systemCSource-2.1**, **systemCSource-2.2**,

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

systemVerilogSource, **systemVerilogSource-3.0**, **systemVerilogSource-3.1**, **systemVerilogSource-3.1a**, **verilogSource**, **verilogSource-95**, **verilogSource-2001**, **vhdlSource**, **vhdlSource-87**, and **vhdlSource-93**

3) **swObject** and **swObjectLibrary**

4) **vhdlBinaryLibrary** and **verilogBinaryLibrary**

5) **executableHdl** and **unelaboratedHdl**

6) **SDC**

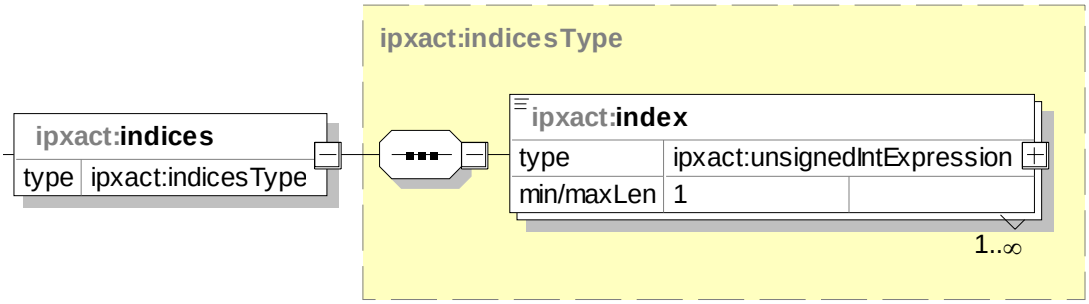
7) **user**

b) **user** (optional; type: ***string***) attribute indicates a user-defined file type name valid only when **fileType** is **user**.

C.9 indices

C.9.1 Schema

The following schema details the information contained in the **indices** element.



C.9.2 Description

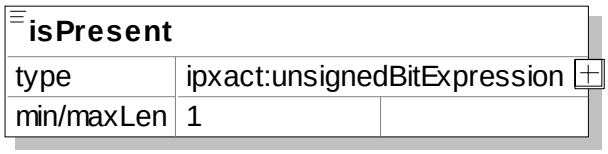
The **indices** element specifies a multi-dimensional list of **index** elements. A **index** (type ***unsignedIntExpression*** (see C.3.7)) is a multi-dimensional array and follows C-semantics for indexing. The **indices** element is typically used to reference into an arrayed/vectored element, such as a port.

C.10 isPresent

C.10.1 Schema

The following schema details the information contained in the **isPresent** element.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



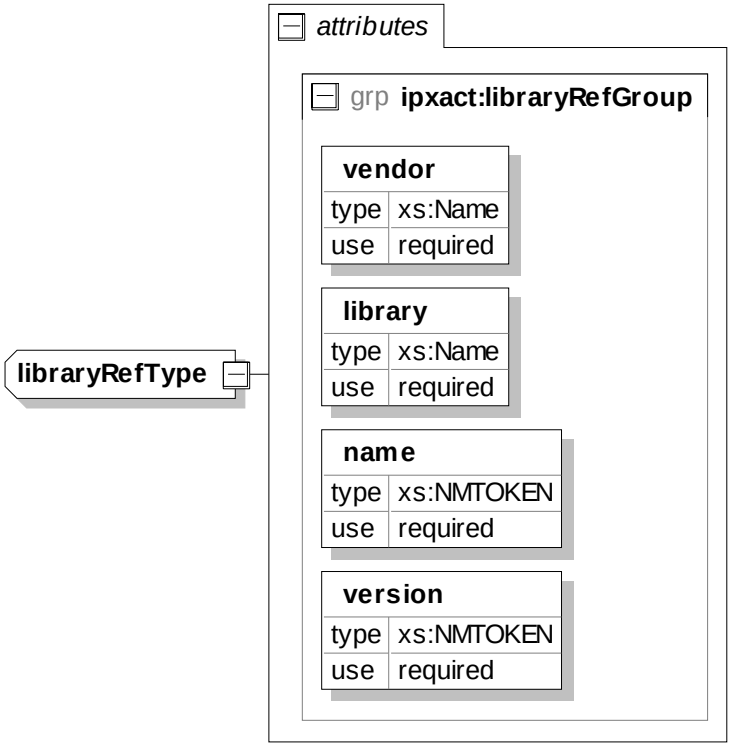
C.10.2 Description

isPresent (type: ***unsignedBitExpression*** (see C.3.5)) specifies an expression that determines how the enclosing element should be treated: *present* (evaluates to `1`) or *nonpresent* (evaluates to `0`). When *present*, the enclosing element shall be treated as described. When *nonpresent*, the enclosing element and all of its descendants shall be treated as if it does not exist in the file.

C.11 libraryRefType

C.11.1 Schema

The following schema details the information contained in the ***libraryRefType*** element.



Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

C.11.2 Description

The ***libraryRefType*** element defines a set of four attributes that reference another IP-XACT description through the unique VLNV identifier.

a) The **vendor** attribute (mandatory; type: ***Name***) identifies the owner of the referenced description.

b) The **library** attribute (mandatory; type: ***Name***) identifies the library of the referenced description.

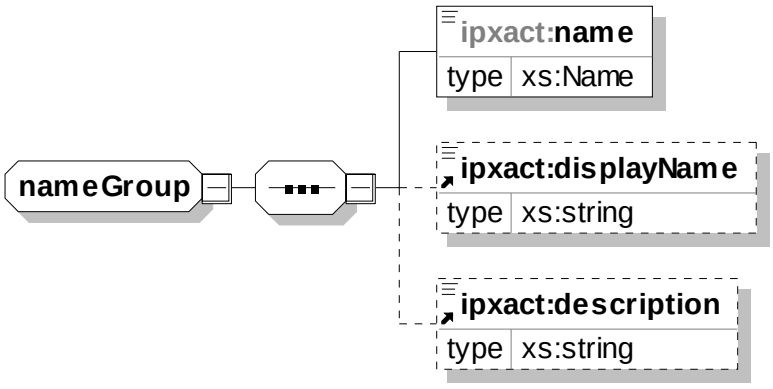
c) The **name** attribute (mandatory; type: ***NMTOKEN***) identifies the name of the referenced description.

d) The **version** attribute (mandatory; type: ***NMTOKEN***) identifies the version of the referenced description.

C.12 nameGroup group

C.12.1 Schema

The following schema details the information contained in the ***nameGroup*** group.



C.12.2 Description

The ***nameGroup*** group defines any descriptive text for the containing element. The ***nameGroup*** group definition contains the following elements.

a) **name** (mandatory; type: ***Name***) identifies the containing element.

b) **displayName** (optional; type: ***string***) allows a short descriptive text to be associated with the containing element.

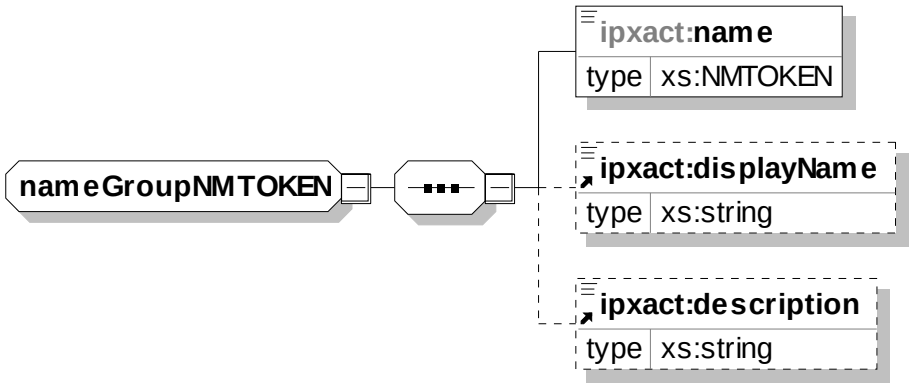
c) **description** (optional; type: ***string***) allows a textual description of the containing element.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

C.13 nameGroupNMTOKEN group

C.13.1 Schema

The following schema details the information contained in the ***nameGroupNMTOKEN*** group.



C.13.2 Description

The ***nameGroupNMTOKEN*** group defines any descriptive text for the containing element. The ***nameGroupNMTOKEN*** group definition contains the following elements.

a) **name** (mandatory; type: ***NMTOKEN***) identifies the containing element. The name used shall match the corresponding port name found in any **view**s of the containing component.

b) **displayName** (optional; type: ***string***) allows a short descriptive text to be associated with the containing element.

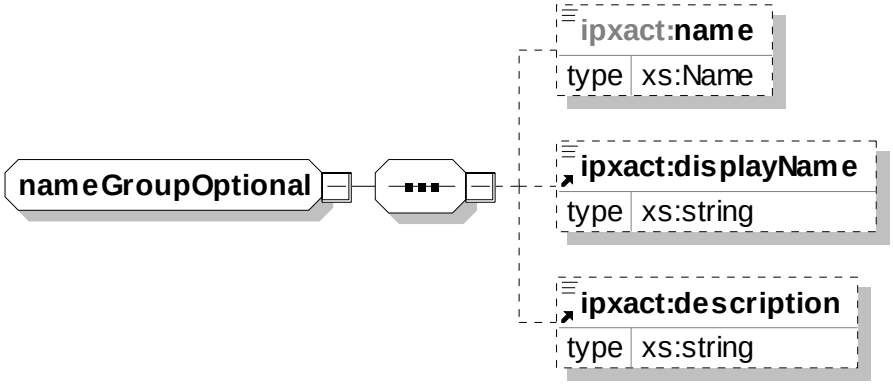
c) **description** (optional; type: ***string***) allows a textual description of the containing element.

C.14 nameGroupOptional group

C.14.1 Schema

The following schema details the information contained in the ***nameGroupOptional*** group.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



C.14.2 Description

The ***nameGroupOptional*** group defines any descriptive text for the containing element. The ***nameGroupOptional*** group definition contains the following elements.

a) **name** ((optional; type: ***Name***) identifies the containing element.

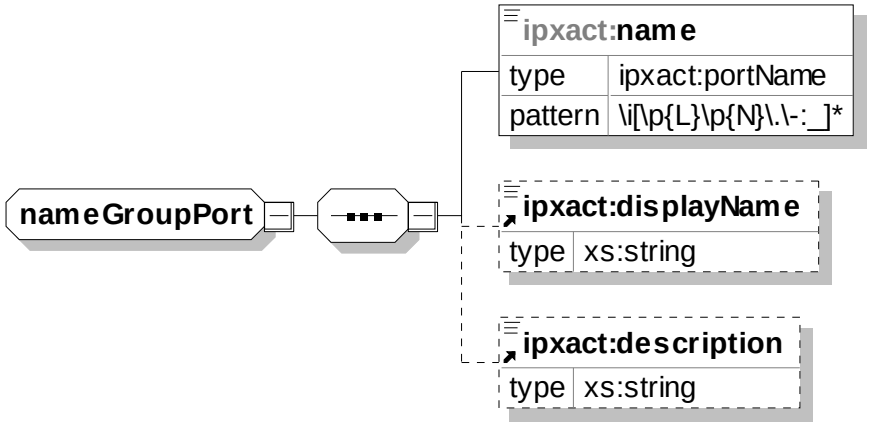
b) **displayName** ((optional; type: ***string***) allows a short descriptive text to be associated with the containing element.

c) **description** ((optional; type: ***string***) allows a textual description of the containing element.

C.15 nameGroupPort group

C.15.1 Schema

The following schema details the information contained in the ***nameGroupPort*** group.



Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

C.15.2 Description

The ***nameGroupPort*** group defines any descriptive text for the containing element. The ***nameGroupPort*** group definition contains the following elements.

a) **name** (mandatory; type: ***portName***) identifies the containing element.

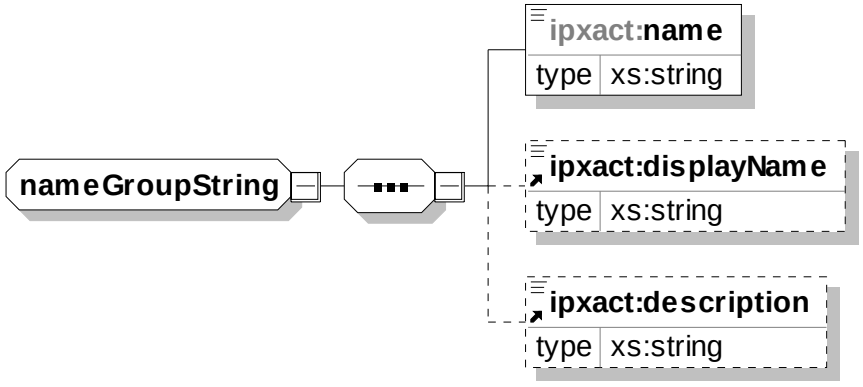
b) **displayName** ((optional; type: ***string***) allows a short descriptive text to be associated with the containing element.

c) **description** ((optional; type: ***string***) allows a textual description of the containing element.

C.16 nameGroupString group

C.16.1 Schema

The following schema details the information contained in the ***nameGroupString*** group.



C.16.2 Description

The ***nameGroupString*** group defines any descriptive text for the containing element. The ***nameGroupString*** group definition contains the following elements.

a) **name** (mandatory; type: ***string***) identifies the containing element.

b) **displayName** (optional; type: ***string***) allows a short descriptive text to be associated with the containing element.

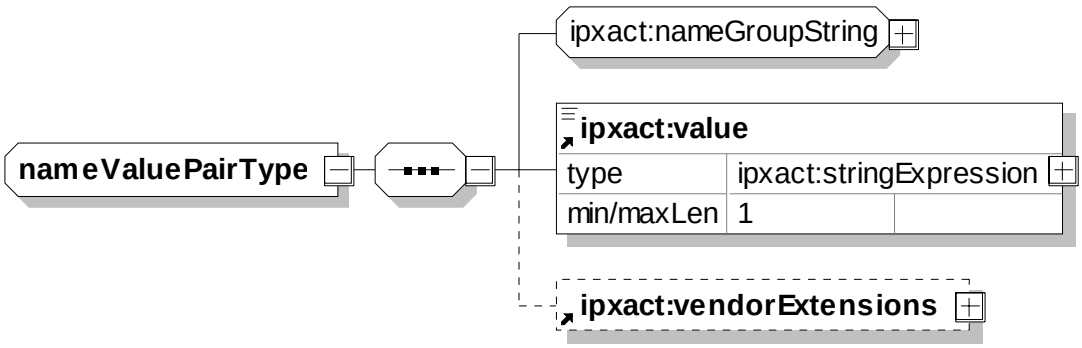
c) **description** (optional; type: ***string***) allows a textual description of the containing element.

C.17 nameValuePairType

C.17.1 Schema

The following schema details the information contained in the ***nameValuePairType*** element.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



C.17.2 Description

The ***nameValuePairType*** element specifies the name and value type used for resolvable elements. The ***nameValuePairType*** element contains the following elements.

a) ***nameGroupString*** group is defined in C.16.

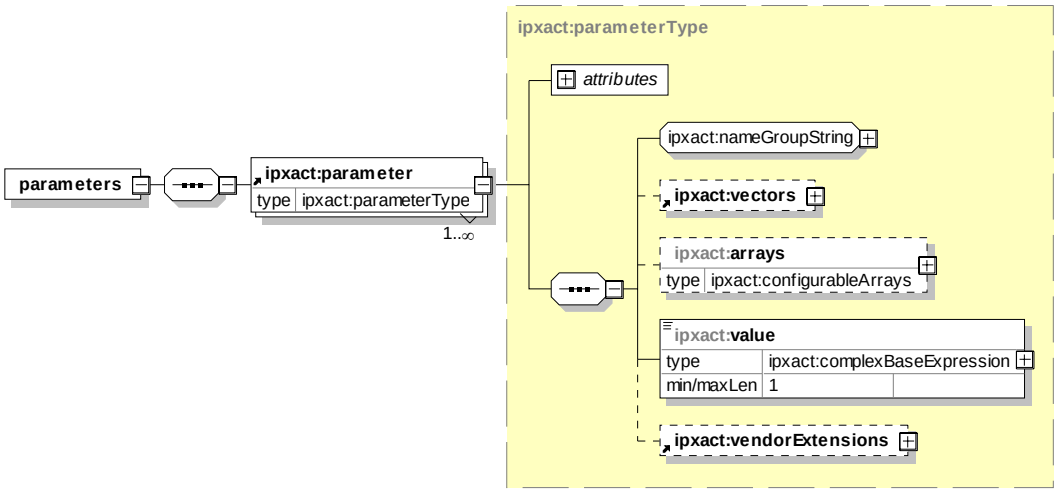
b) **value** (mandatory; type: ***stringExpression*** (see C.3.3)) contains the actual value of the **parameter**.

c) **vendorExtensions** (optional) adds any extra vendor-specific data related to the port. See C.24.

C.18 parameters

C.18.1 Schema

The following schema details the information contained in the **parameters** element.



Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

C.18.2 Description

The **parameters** element contains an unbounded list of **parameter** elements. **parameter** (mandatory) defines a configurable element related to the containing element. The parameter definition allows for the assignment of a name, **parameterID**, and a value. The **parameter** element also allows for vendor attributes to be applied. The **parameter** element definition contains the following elements.

a) ***attributes*** specify the parameter behavior, like **type**, **sign**, **parameterId**, etc. See C.19.

b) ***nameGroupString*** is defined in C.16.

c) **vectors** (optional) specifies the dimensions of a vector when the parameter represents a packed array or bit-vector (where the **type** is `bit`). See C.23.

d) **arrays** (optional; type: ***configurableArrays*** (see C.4)) specifies dimensions for a port defined as an array.

e) **value** (mandatory; type: ***complexBaseExpression*** (see C.3)) contains the actual value of the **parameter**. The **value** element is further constrained by the **type**, **sign**, and **array** properties specified on the **parameter** element.

f) **vendorExtensions** (optional) adds any extra vendor-specific data related to the **parameter**. See C.24.

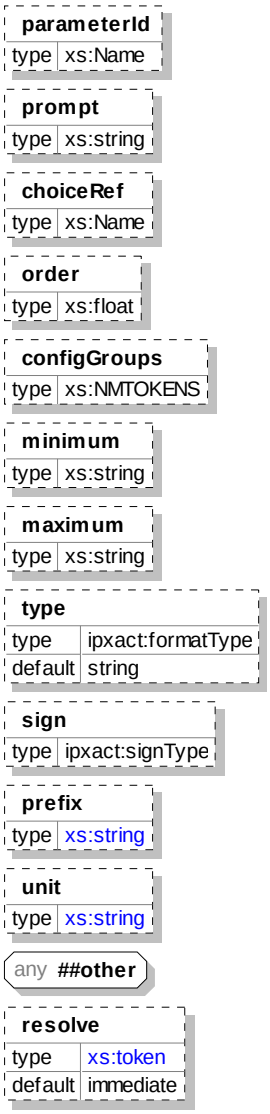
See also: SCR 5.2, SCR 5.16, SCR 5.20, SCR 5.21, and SCR 5.22.

C.19 attributes

C.19.1 Schema

The following schema details the information contained in the ***attributes*** for various elements.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.



C.19.2 Description

The **parameter** attributes specify the parameter behavior via the following elements.

a) The **parameterid** attribute (optional; type: ***Name***) assigns a unique identifier to the containing parameter for reference throughout the containing description. **parameterid** is required when the element has a **resolve** type equal to **user** or **generated**, or is referenced from an expression. This **parameterid** can be referenced in two ways: by the **configurableElementValue** element (see C.5.2) in the referencing description or in an expression.

b) The **prompt** attribute (optional; type: ***string***) defines a prompt string that a DE can use if the **resolve** attribute is equal to **user**.

c) The **choiceRef** attribute (optional; type: ***Name***) indicates the value of the containing element is defined in the referenced **choice** element.

d) The **order** attribute (optional; type: ***float***) indicates how elements are presented, when **resolve** equals **user**. The elements are presented in ascending order.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

e) The **configGroups** attribute (optional; type: ***NMTOKENS***) indicates a name to group elements together; elements with matching values for this attribute are contained in the same group. There is no semantic meaning to this attribute.

f) The **minimum** attribute (optional; type: ***string***) indicates the lower bound for the **value** of the containing element. This check is only valid for a format of **byte**, **shortint**, **int**, **longint**, **real**, or **shortreal**. The **type** attribute shall specify the type of the **minimum** attribute.

g) The **maximum** attribute (optional; type: ***string***) indicates the lower bound for the **value** of the containing element. This check is only valid for a format of **byte**, **shortint**, **int**, **longint**, **real**, or **shortreal**. The **type** attribute shall specify the type of the **maximum** attribute.

h) The **type** attribute (optional; default: **string**) is the type to which the parameter value resolves. The value shall be one of the following.

1) **bit** indicates the value shall resolve to a SystemVerilog `bit` which by default is resolved to a 1-bit value, unless a vector size has been specified.

2) **byte** indicates the value shall resolve to a SystemVerilog `byte` which is resolved to an 8-bit integer value.

3) **shortint** indicates the value shall resolve to a SystemVerilog `shortint` which is resolved to a 16-bit integer value.

4) **int** indicates the value shall resolve to a SystemVerilog `int` which is resolved to a 32-bit integer value.

5) **longint** indicates the value shall resolve to a SystemVerilog `longint` which is resolved to a 64-bit integer value.

6) **shortreal** indicates the value shall resolve to a SystemVerilog `shortreal` which is resolved to a 32-bit floating point value.

7) **real** indicates the value shall resolve to a SystemVerilog `real` which is resolved to a 64-bit floating point value.

8) **string** indicates the value shall resolve to a SystemVerilog `string`.

i) The **sign** attribute (optional) indicates the signed-ness of the parameter value. This property only influences the following types: **bit**, **byte**, **shortint**, **int**, and **longint**:

where the types **byte**, **shortint**, **int**, and **longint** are `signed` by default
and the type **bit** is `unsigned` by default.

The **sign** property does not influence **real**, **shortreal**, **bit**, and **string** types; effectively, **sign** can be ignored for these types.

j) The **prefix** attribute (optional; type: ***string***) specifies the prefix that precedes the **unit** of a value. The **prefix** is for informational purposes only and is not applied to the value (e.g., when evaluating expressions). Its possible values are:, **hecto**, **kilo**, **mega**, **giga**, **tera**, **peta**, **exa**, **zetta**, **yotta**, **deci**, **centi**, **milli**, **micro**, **nano**, **pico**, **femto**, **atto**, **zepto**, and **yocto**. These represent the values as defined by the *International System of Units* [B14].

k) The **unit** attribute (optional; type: ***string***) specifies the unit of a value. The **unit** is for informational purposes only and is not applied to the value (e.g., when evaluating expressions). Its possible values are: **second**, **ampere**, **kelvin**, **hertz**, **joule**, **watt**, **coulomb**, **volt**, **farad**, **ohm**, **siemens**, **henry**, and **Celsius**. These represent some of the base units and derived units as defined by the *International System of Units* [B14].

l) The **any ##other** attribute (optional) indicates any additional attributes in other name spaces are allowed in the containing element. These additional attributes are called *vendor attributes*.

m) The **resolve** attribute (optional; type: ***token***; default: **immediate**) defines how the value for the containing element is configured. The value shall be one from the enumerated list of **immediate**, **user**, or **generated**.

1) **immediate** indicates the value shall be specified in the containing element.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

2) **user** indicates the value shall be specified by user input and the new value stored in a referencing description under the **configurableElementValue** element (see C.5.2).

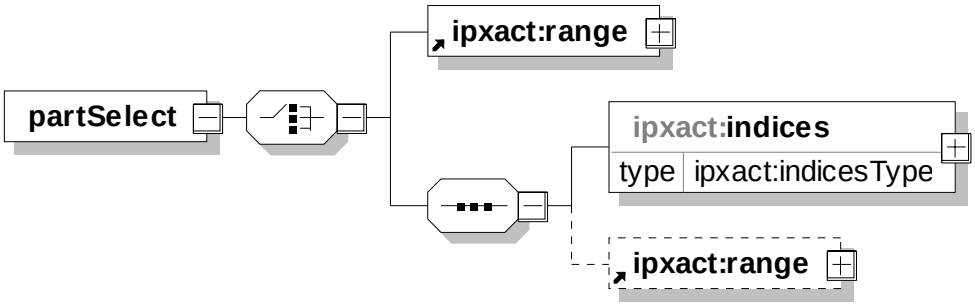
3) **generated** indicates the value shall be set by a generator and the new value stored in a referencing description under the **configurableElementValue** element (see C.5.2).

See also: SCR 5.1.

C.20 partSelect

C.20.1 Schema

The following schema details the information contained in the **partSelect** element.



C.20.2 Description

The **partSelect** element defines a selection of sub-elements from within a referenced port element. A part selection can consist of either a single range specification or multiple index specifications along with an optional range specification.

a) **range** defines the range of bits or array elements being referenced. If **index** elements are also present, the **range** applies after all **index** elements. See C.22.

b) **indices** specifies a list of **index** elements. The **indices** specify an element in the IP-XACT object to which this **accessHandle** applies. See C.9.

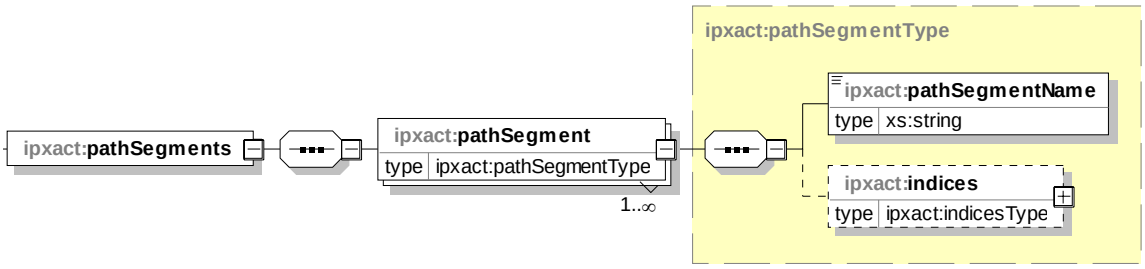
When multiple indices are present, they correspond to the dimensions of the defined port, account for array ranges first, and then vector ranges.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

C.21 pathSegments

C.21.1 Schema

The following schema details the information contained in the **pathSegments** element.



C.21.2 Description

The **pathSegments** element specifies an ordered list of **pathSegment** elements. A **pathSegment** is one node in the hierarchical path. When concatenated with a desired separator, the elements in this form a language specific path for the parent slice into the referenced view. The **pathSegment** element contains the following elements.

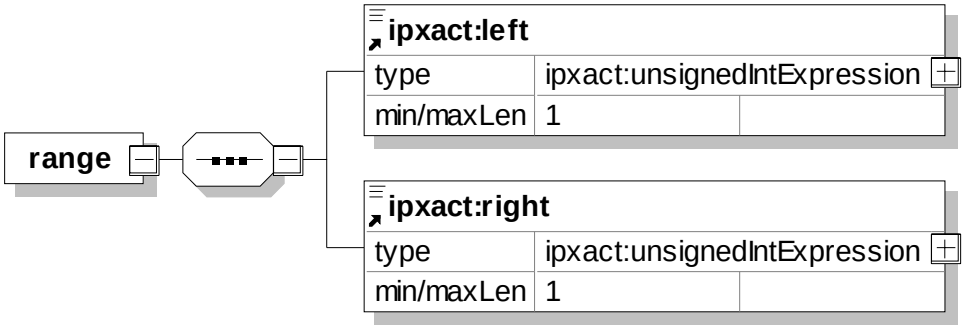
a) **pathSegmentName** (mandatory; type: ***string***) is one node in the path.

b) **indices** (optional) specifies a list of **index** elements. The **indices** specify an element in the IP-XACT object to which this **accessHandle** applies. See C.9.

C.22 range

C.22.1 Schema

The following schema details the information contained in the **range** element.



Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

C.22.2 Description

The **range** element defines the range of bits or array elements being referenced. The **range** element contains the following elements.

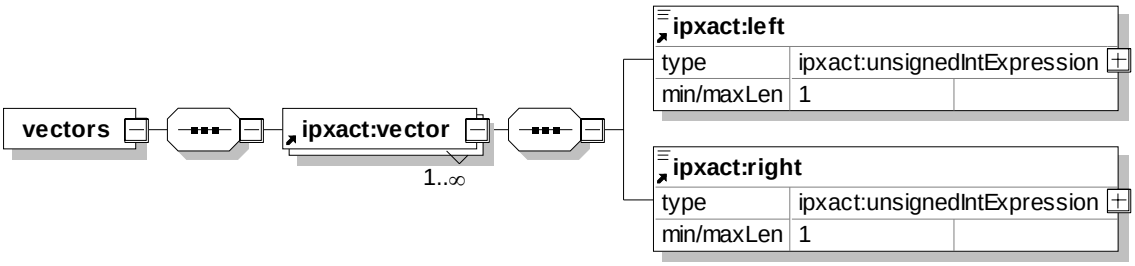
a) **left** (mandatory; type: ***unsignedIntExpression*** (see C.3.8)) specifies the left range of the bit slice.

b) **right** (mandatory; type: ***unsignedIntExpression*** (see C.3.8)) specifies the right range of the bit slice.

C.23 vectors

C.23.1 Schema

The following schema details the information contained in the **vectors** element.



The **vectors** element specifies the dimensions of a **vector** when the parameter represents a packed array or bit-vector (where the **type** is `bit`). The **vector** element contains the following elements.

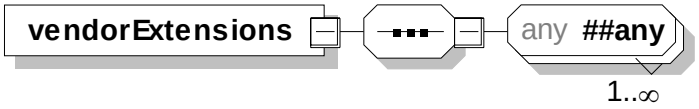
a) **left** (mandatory; type: ***unsignedIntExpression*** (see C.3.8)) specifies the left range for the bit slice used to map a port vector to the bus interface.

b) **right** (mandatory; type: ***unsignedIntExpression*** (see C.3.8)) specifies the right range for the bit slice used to map a port vector to the bus interface.

C.24 vendorExtensions

C.24.1 Schema

The following schema details the information contained in the **vendorExtensions** element.



C.24.2 Description

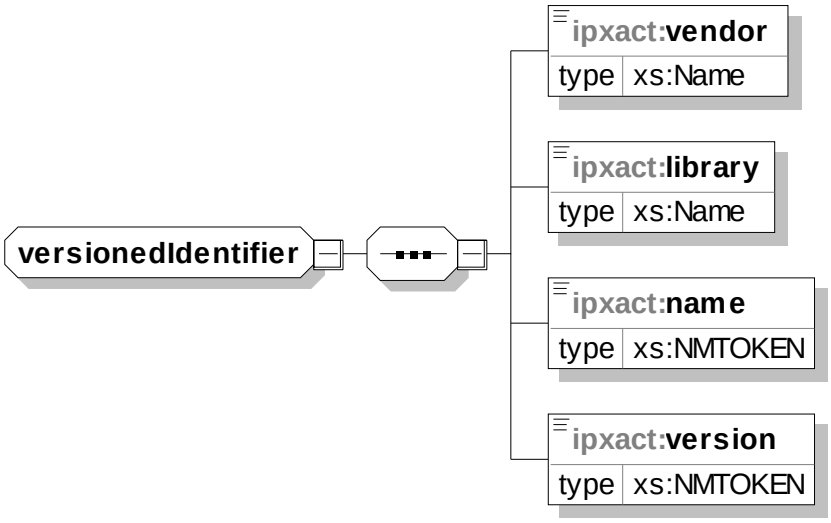
The **vendorExtensions** element is a place in the description in which any vendor specific information can be stored. The **vendorExtensions** element allows any well-formed description.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

C.25 versionedIdentifier group

C.25.1 Schema

The following schema details the information contained in the ***versionedIdentifier*** group.



C.25.2 Description

The ***versionedIdentifier*** group defines a unique reference of or from an IP-XACT description. Only one object with a given VLNV may be present in a DE at any given time. The timing and way to change the VLNV of an object is completely up to the user or developer. The ***versionedIdentifier*** group definition contains the following four subelements.

a) **vendor** (mandatory; type: ***Name***) identifies the owner of this description. The format of the **vendor** element is the company internet domain name in left-to-right order (e.g., `accellera.org` not `org.accellera`).

b) **library** (mandatory; type: ***Name***) identifies the library of this description. This allows a vendor to group descriptions.

c) **name** (mandatory; type: ***NMTOKEN***) identifies the name of this description within a library.

d) **version** (mandatory; type: ***NMTOKEN***) identifies the version of this description. This allows a vendor to provide many descriptions that all have the same name, but are still uniquely identified. The **version** may appear as an alphanumeric string and contain a set of substrings, with non-alphanumeric delimiters in-between. Each IP supplier shall have their own cataloguing system for setting version numbers.

See also: SCR 1.1 and SCR 1.2.

C.25.3 Sorting and comparing version elements

Sorting and comparing **version** elements determines whether:

— an IP is a component that has been previously imported;

— multiple versions of the same IP exist in a design;

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

— a newer version of an IP exists.

To sort and compare **version** elements, subdivide the version number into major fields and subfields. Major fields may be separated by a non-alphanumeric delimiter such as:, `.`, `-`, _, etc. Each major field can be compared to determine equivalence and broken down further into subfields if necessary.

C.25.3.1 Comparison rules

a) Each **version** element is broken into its major fields, which are separated using the appropriate delimiter (e.g., `:` or `.`).

b) Major fields are compared against each other from left to right.

c) Subfields, within each major field, need to be examined if the major fields are alphanumeric. Each major field shall have alphabetical and numerical subfields that are separated from right to left.

d) To summarize the rules for the comparison of each subfield in a major field:

   1) Numeric—Compare the integer values of numeric subfields.

   2) Alphabetic

      i) String—Perform a simple string comparison.

      ii) Case—Ignore alphabetic case (e.g., `a` and `A` are the same).

It is possible for different representations of version numbers to be considered equal. For example, under these rules, `A1` and `A01` are equal, since numerical subfields are compared numerically, and `A.B` equals `A_B`, since delimiters are not compared.

C.25.3.2 Comparison examples

The following examples illustrate the sorting and comparing of a **version** elements.

Example 1

The first case uses: `205:75WR16` and `215:50HR15`.

a) Each of these version numbers break down into the following two major fields, separated by the `:` delimiter: `205 75WR16` and `215 50HR15`.

b) Major fields are compared against each other from left to right. In this example, the first major fields (`205` and `215`) differ between the VLNV strings and the comparison ends there. This case is also simplified since the first major field is an integer (i.e., numeric).

c) Subfields, within each major field, need to be examined if the major fields are alphanumeric. Each major field shall have alphabetical and numerical subfields that are separated from right to left.

Example 2

In the next case, two VLNV have the same first major field, and their second major subfields need to be compared: e.g., `205:45R16` and `205:55R15`.

a) The first major field (`205`) is the same between these two VLNV, so the second major field is checked. These second major fields are broken down into the following alphabetic and numeric subfields: `45 R 16` and `55 R 15`.

b) The subfields are compared from left to right. The first (and in this case only) comparison is `45` versus `55`, so these subfields are not equal. The major fields are not equivalent.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

C.25.4 Version control

Each file conforming to the top-level schema has a set of VLNV elements that, when considered together, form a unique identifier (a *version control number*) for the information contained in that XML document. The VLNV of any IP-XACT information is not the same as the version of the file that might contain that information.

NOTE—An XML file might be revised in a way that does not materially affect the IP-XACT information content. For example, copyright notices are updated, comments are added, and environment variable names used as part of the filenames might be changed (but still point to the same files). These changes may not necessitate changing the VLNV.

Many developers of IP libraries use a version control system to track updates and changes to the various files that contribute to the overall design and IP package information. At any time, individual files may be modified and updated as development of that design or IP progresses. At appropriate junctures, releases are made, each consisting of a particular combination of files at different levels of a version.

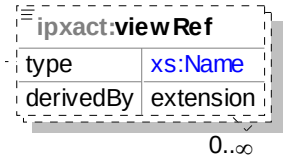
An IP-XACT description is one of the files that can be very usefully tracked in this way and updated in line with other design modifications. There is no direct link between the version number of the file and the VLNV identifier contained in that description. In many cases, however, the VLNV can be coordinated with the overall release package version.

See also: SCR 1.1 and SCR 1.2.

C.26 viewRef

C.26.1 Schema

The following schema details the information contained in the **viewRef** element.



C.26.2 Description

The occurrence of **viewRef** (type: ***Name***) elements (`0` to *infinity*) in an **accessHandle**, **abstractionType**, **wireTypeDef**, and **transTypeDef** elements shall meet the following rules and interpretation.

- If an **accessHandle**, **abstractionType**, **wireTypeDef**, or **transTypeDef** element does not have **viewRef** elements, then all existing component view elements are referenced implicitly. This implies there is only one **accessHandle**, **abstractionType**, **wireTypeDef**, or **transTypeDef** element in the enclosing **accessHandle**s, **abstractionType**s, **wireTypeDef**s, and **transTypeDef**s elements.
- A view name may be referenced once in all **viewRef** elements of the enclosing **accessHandle**s, **abstractionType**s, **wireTypeDef**s, or **transTypeDef**s elements.
- If a view name is not explicitly or implicitly referenced in an **accessHandle**s, **abstractionType**s, **wireTypeDef**s, or **transTypeDef**s elements, the enclosing element of the **accessHandle**s, **abstractionType**s, **wireTypeDef**s, or **transTypeDef**s element does not exist in that view.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

C.27 xml:id

The **xml:id** attribute can be specified to provide a unique identifier for the element. Refer to *xml:id Version 1.0 [xml-ref]*. The attribute value is required to be unique in the document. The attribute is not referenced by any of the features in the IP-XACT language. **xml:id** has been provided so tools can more precisely identify nodes in the document to provide, e.g., better change control or to attach documentation and other related information to the node in the document.

xml:id is available on all nodes which otherwise can not be uniquely identified, i.e., all items which can have multiple occurrences at the same level of the hierarchy.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Annex D

(normative)

Types

Many elements and attributes defined in the standard have associated types. These types define the legal values and ranges for input into these element and attributes.

D.1 boolean

The **boolean** type defines two possible values, **true** and **false**.

D.2 float

The **float** type defines a decimal floating point number based on the IEEE single-precision 32-bit floating point type (see IEEE Std 754-1985 [B2]).

D.3 ID or IDREF

The **ID** or **IDREF** type defines a unique identifier through the containing description. It needs to begin with a letter or underscore (_). An **ID** or **IDREF** shall only contain letters, numbers, and the underscore (_), dash (`-`), and dot (`.`) characters. Any leading or trailing spaces are removed.

D.4 instancePath

The **instancePath** type defines a series of **Name** type character strings, see D.7, separated by a slash (`/`). Any leading or trailing space is removed.

D.5 integer

The **integer** type defines a decimal integer number of infinite precision, containing the numbers `0`–`9`.

D.6 libraryRefType

The **libraryRefType** type is an element type, not a data type. This type defines the four attributes of a VLNV required for a reference from one description to another description. See C.11.

D.7 Name

The **Name** type defines a series of any characters, excluding embedded whitespace. It needs to begin with a letter, colon (`:`), or underscore (_). A **Name** shall only contain letters, numbers, and the colon (`:`), underscore (_), dash (`-`), and dot (`.`) characters. Any leading or trailing spaces are removed.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

D.8 NMTOKEN

The **NMTOKEN** type defines a series of any characters, excluding embedded whitespace. It shall only contain letters, numbers, and the colon (`:`), underscore (_), dash (`-`), and dot (`.`) characters. Any leading or trailing spaces are removed.

D.9 NMTOKENS

The **NMTOKENS** type defines a series of any characters, including embedded whitespace. It shall only contain letters, numbers, and the colon (`:`), underscore (_), dash (`-`), and dot (`.`) characters.

D.10 portName

The **portName** type defines a series of any characters, excluding embedded whitespace. It shall only contain letters, numbers, and the colon (`:`), underscore (_), dash (`-`), and dot (`.`) characters. It also needs to begin with a letter, colon (`:`), or underscore (_). Any leading or trailing spaces are removed.

D.11 ipxactURI

The **ipxactURI** type defines a string of characters for an absolute or relative path to a file, a directory, or an executable in URI format (`xs:anyURI`), except it can contain environment variables in the `${ENV_VAR}` form, which are replaced by their value(s) to provide the underlying URI.

D.12 string

The **string** type defines a series of any characters and may include spaces.

D.13 token

The **token** type defines a series of any characters, excluding carriage-return, line-feed, and tab. Any leading or trailing spaces are removed and all internal sequences of two or more spaces are reduced to one space.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Annex E

(normative)

SystemVerilog expressions

This standard utilizes *SystemVerilog expressions* as a means to specify an equation as the value of an element. Expressions can be specified for all predefined element values having an associated type (see C.3 and C.18) which allows for an expression to be specified. Besides predefined types, parameters can also be specified using expressions, in which case the type attributes define the type of expression which can be specified.

E.1 Overview

The IP-XACT Expression Language is based on the SystemVerilog Expression Language as specified in the IEEE 1800-2012 Standard (for SystemVerilog-Unified Hardware Design, Specification, and Verification Language).

The goal of the IP-XACT Expression Language is for it to be a subset of IEEE 1800-2012, i.e., the expressions defined in IP-XACT should be easily ported-to or incorporated-into a SystemVerilog file and interpreted by any SystemVerilog parser/processor.

The IP-XACT Expression Language, however, should also be able to represent constructs and functionality specific to the IP-XACT language. To make the language more in-line with the IP-XACT language, some functionality has been limited and some changes to the original SystemVerilog Expression Language were introduced.

E.2 Data-types

A subset of the SystemVerilog data-types are used by the IP-XACT Expression Language. All IP-XACT data-types map to a specific SystemVerilog data-type. The casting functionality and operators follow the same subset of the functionality specified by the SystemVerilog Expression Language.

It has been decided to not support complex, user-defined, and time data-types in IP-XACT; it has also been decided to only support 2-state values and not allow for unknown or high-impedance values. The supported data-types are shown in Table E1.

Table E1—Data-types

Data-type	Description
bit	2-state data type, user-defined vector size
byte	2-state data type, 8-bit signed integer
shortint	2-state data type, 16-bit signed integer
int	2-state data type, 32-bit signed integer
longint	2-state data type, 64-bit signed integer

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table E1—Data-types *(continued)*

shortreal	32-bit floating point
real	64-bit floating point
string	An ordered collection of characters

E.2.1 bit data type

The **bit** data-type is equivalent to the **bit** data-type in SystemVerilog.

E.2.2 byte data type

The **byte** data-type is equivalent to the **byte** data-type in SystemVerilog, except it is not possible to specify an ASCII character and, therefore, ASCII characters cannot be assigned to a **byte** variable.

E.2.3 shortint data type

The **shortint** data-type is equivalent to the **shortint** data-type in SystemVerilog.

E.2.4 int data type

The **int** data-type is equivalent to the **int** data-type in SystemVerilog.

E.2.5 longint data type

The **longint** data-type is equivalent to the **longint** data-type in SystemVerilog.

E.2.6 shortreal data type

The **shortreal** data-type is equivalent to the **shortreal** data-type in SystemVerilog.

E.2.7 real data type

The **real** data-type is equivalent to the **real** data-type in SystemVerilog.

E.2.8 string data type

The IP-XACT Expression Language **string** data-type is equivalent to the SystemVerilog **string** datatype; just like the SystemVerilog **string,** it only supports US-ASCII characters.

However, to simplify the string functionality and avoid potential issues with string-length, indexing, and casting to and from **long** or **bitstring** values; casting, indexing, and length tests are not supported. This means an IP-XACT **string** can be seen as an immutable, 'simple' object without length, which cannot be cast to other types or from other types. This also means only the string-equality, inequality, concatenation, replication, and inside operators are supported (see Table E.2).

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

E.2.9 Signed and unsigned data types

The IP-XACT data-types follow the same default signed/unsigned functionality as has been defined in the SystemVerilog Expression Language, i.e., the **bit** data-type is unsigned and the **byte**, **shortint**, **int**, and **longint** data-types are signed by default.

IP-XACT also allows specifying the signed-ness when assigning the values. and just like in SystemVerilog, the signed-ness can also be used when casting the value.

E.3 Assignments

Variables and constants are fully declared and assigned using IP-XACT parameters (see C.18). This means the IP-XACT Expression Language itself does not support assignments and, therefore, does not support any assignment operators.

E.3.1 Single value assignment

To assign a single parameter with an identifier of `baseAddress` having a value of `'h100` in IP-XACT, use the following format:

```
<ipxact:parameter type="longint" parameterId=”baseAddress”>
  <ipxact:name>baseAddress</IP-XACT:name>
  <ipxact:value>'h100</IP-XACT:value>
</ipxact:parameter>
```

which is equivalent to the following SystemVerilog assignment:

```
longint baseAddress = 'h100;
```

E.3.2 Parameter type

The value of a parameter is itself an expression and should be specified accordingly. The type of the parameter expression is defined by the type attribute. Possible values for this type attribute are: bit, byte, shortint, int, longint, shortreal, real, string. These types correspond to their SystemVerilog counterparts as described in E.2.

The expression that is specified as the parameter’s value is implicitly cast to the parameter’s type. See C.3 and C.18.

E.3.3 Parameter signing

To change the signed-ness of a parameter, use the **sign** attribute (see C.19.2), whose value can be **signed** or **unsigned**, e.g.,

```
<ipxact:parameter type="longint" parameterId=”baseAddress” sign=”signed”>
  <ipxact:name>baseAddress</IPXACT:name>
  <ipxact:value>'h100</IPXACT:value>
</ipxact:parameter>
```

This property only influences the following types: **bit**, **byte**, **shortint**, **int**, and **longint**:
where the types **byte**, **shortint**, **int**, and **longint** are `signed` by default
and the type **bit** is `unsigned` by default.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

The **sign** property does not influence **real**, **shortreal**, **bit**, and **string** types; effectively, **sign** can be ignored for these types.

E.3.4 Vector assignment

To create a packed array bit-string, the IP-XACT Expression Language allows the specification of vector information. Vector information can only be specified for parameters with a type of **bit**.

The following shows how to create a 8-bit packed array parameter with an identifier of `test` having the value `8'hAB`.

```
<parameter type="bit" parameterId=”test”>
  <name>test</name>
  <vectors>
    <vector>
      <left>7</left><right>0</right>
    </vector>
  </vectors>
  <value>8'hAB</value>
</parameter>
```

Which is the equivalent of

```
bit [7:0] test = 8'hAB;
```

E.3.5 Array assignment

To create an array for any type of parameter, the IP-XACT Expression Language allows the specification of array information and multiple initial values.

The following shows how to create an array parameter with a type of **int** and an identifier of `test` having the value `'{8'hAA,8'hAA,8'hAA,8'hAA,8'hAA,8'hAA,8'hAA}`.

```
<parameter type="int" parameterId=”test”>
  <name>test</name>
  <arrays>
    <array>
      <left>6</left><right>0</right>
    </array>
  </arrays>
  <value>'{8'hAA, 8'hAA, 8'hAA, 8'hAA, 8'hAA, 8'hAA, 8'hAA}</value>
</parameter>
```

This is the equivalent of

```
longint test [0:6] = '{8'hAA,8'hAA,8'hAA,8'hAA,8'hAA,8'hAA,8'hAA};
```

The following shows how to create a packed bit array parameter with an identifier of `test` having the array value `{'{8'h0,8'h1,8'h2,8'h3,8'h4,8'h5},'{8'h6,8'h7,8'h8,8'h9,8'hA,8'hB}}`.

```
<parameter type="bit" parameterId=”test”>
  <name>test</name>
  <vectors>
    <vector>
      <left>7</left><right>0</right>
```


Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
      </vector>
    </vectors>
    <arrays>
      <array>
        <left>1</left><right>0</right>
        <left>5</left><right>0</right>
      </array>
    </arrays>
    <value>'{'{8'h0,8'h1,8'h2,8'h3,8'h4,8'h5},
     '{8'h6,8'h7,8'h8,8'h9,8'hA,8'hB}}</value>
  </parameter>
```

This is the equivalent of

```
bit [0:7] test [0:1] [0:5] =
    '{'{8'h0,8'h1,8'h2,8'h3,8'h4,8'h5},'{8'h6,8'h7,8'h8,8'h9,8'hA,8'hB}};
```

E.3.6 Identifiers

The **parameterId** (see C.19.2), which is used as the identifier for the parameter by the expression language, can be specified using all characters from the *XML schema NameChar definition*; however, the first character of the **parameterId** shall not be a `-`, `:`, or `.`, a number, or any other character as specified by the *XML schema NameStartChar definition*.

The **parameterId** shall be unique within the containing document.

E.3.7 Identifier references

Parameters can be used in other expressions by referencing their identifier, the **parameterId**.

```
<ipxact:parameter parameterId="offset">
<ipxact:name>offset</ipxact:name>
<ipxact:value>'h100</ipxact:addressOffset>
</ipxact:parameter>
<ipxact:register>
…
<ipxact:addressOffset>offset + 'h10</ipxact:addressOffset>
…
</ipxact:register>
```

Any parameter in the document with a **parameterId** can be referenced from an expression. However, view-dependent parameters specified inside the instantiation sections cannot be referenced from outside the instantiation which specifies the parameter.

The **parameterId** can be specified using the characters `-`, `:`, or `..` However, to avoid confusion with the expression operators, these characters can not be directly specified when referencing the parameter from an expression. In that case, these characters should be escaped first (see E.3.7.1).

Example

```
shiftreg_a
busa_index
error_condition
merge_ab
_bus3
```

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Implementations may set a limit on the maximum length of identifiers, but the limit shall be at least 1024 characters. If an identifier exceeds the implementation-specific length limit, an error shall be reported.

E.3.7.1 Escaped identifier references

Escaped identifier references shall start with the backslash character (\) and end with white space (a space, tab, or newline). They provide a means of including the characters which are part of the *XML schema NameChar definition*, but are not allowed to be specified directly for an identifier.

The following extra characters can be used for an escaped character: `-`, `:`, and `.`. Neither the leading backslash character nor the terminating white space is considered to be part of the identifier reference. Therefore, the escaped identifier reference `\cpu3` is treated the same as the non-escaped identifier references `cpu3`.

Example

```
\busa:index
\:clock
\error-condition
\a.b
```

E.3.7.2 Keywords

Keywords are predefined non-escaped identifiers that are used to define the language constructs. All keywords are defined in lowercase only. A keyword preceded by an escape character (see E.3.7.1) is not interpreted as a keyword.

A subset of SystemVerilog keywords have been specified for IP-XACT. Since fewer keywords have been specified in the IP-XACT Expression Language, the other SystemVerilog keywords need to be escaped (see E.3.7.1) before processing these expressions using a SystemVerilog parser/processor.

The following keywords are reserved: **bit**, **byte**, **shortint**, **int**, **longint**, **shortreal**, **real**, **signed**, **unsigned**, **inside**, and **with**.

E.4 Operators

Table E.2 gives an overview of the SystemVerilog operators and how IP-XACT supports then (or not).

Table E.2—SystemVerilog operators

Operator token	Name	Operand data types
=	Binary assignment operator	Assignments are not supported
+= -= /= *=	Binary arithmetic assignment operators	Assignments are not supported
%=	Binary arithmetic modulus assignment operator	Assignments are not supported
&= |= ^=	Binary bit-wise assignment operators	Assignments are not supported
>>= <<=	Binary logical shift assignment operators	Assignments are not supported
>>>= <<<=	Binary arithmetic shift assignment operators	Assignments are not supported
?:	Conditional operator	integral[1], real[2], string

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table E.2—SystemVerilog operators *(continued)*

+ -	Unary arithmetic operators	integral[1], real[2]
!	Unary logical negation operator	integral [1], real[2]
~ & ~& | ~| ^ ~^ ^~	Unary logical reduction operators	integral [1]
+ - * / **	Binary arithmetic operators	integral [1], real[2]
%	Binary arithmetic modulus operator	integral [1]
& | ^ ^~ ~^	Binary bit-wise operators	integral [1]
>> <<	Binary logical shift operators	integral [1]
>>> <<<	Binary arithmetic shift operators	integral [1]
&& || -> <->	Binary logical operators	integral [1], real[2]
< <= > >=	Binary relational operators	integral [1], real[2]
=== !==	Binary case equality operators	Unknown or high-impedance values are not supported
== !=	Binary logical equality operators	integral [1], real[2], string
==? !=?	Binary wildcard equality operators	Unknown or high-impedance values are not supported
++ --	Unary increment and decrement operators	Assignments are not supported
inside	Binary set membership operator	integral [1], real[2], string
dist	Binary distribution operator	Randomization is not supported
{} {{}}	Concatenation and replication operators	integral[1], string
{<<{}} {>>{}}	Stream operators	integral[1]

[1] integral = **bit**, **byte**, **shortint**, **int**, and **longint**
[2] real = **real** and **shortreal**

All operators are supported except:

a) The assignment operators are not supported, since assignments are handled using IP-XACT constructs.

b) Operators which handle high-impedance and unknown equality are not supported. The IP-XACT Expression Language does not specify data-types which support high-impedance and unknown values, which means there is no need to support these operators.

c) The binary distribution operator is not supported because it is not possible to specify randomized values.

E.5 Math functions

The SystemVerilog Expression Language supports a set of math functions; to make the expression language more complete, the exact same set of functions is supported.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

E.5.1 Integer math function

The integer math function is shown in Table E.3.

Table E.3—Integers

Function	Description
$clog2(x)	Returns the ceiling of the log base 2 of the argument (the log rounded up to an integer value)

E.5.2 Real math functions

The real math functions are shown in Table E.4.

Table E.4—Reals

Function	Description
$ln(x)	Natural logarithm
$log10(x)	Decimal logarithm
$exp(x)	Exponential
$sqrt(x)	Square root
$pow(x,y)	x**y
$floor(x)	Floor
$ceil(x)	Ceiling
$sin(x)	Sine
$cos(x)	Cosine
$tan(x)	Tangent
$asin(x)	Arc-sine
$acos(x)	Arc-cosine
$atan(x)	Arc-tangent
$atan2(y,x)	Arc-tangent of y/x
$hypot(x,y)	sqrt(x*x+y*y)
$sinh(x)	Hyperbolic sine
$cosh(x)	Hyperbolic cosine
$tanh(x)	Hyperbolic tangent
$asinh(x)	Arc-hyperbolic sine
$acosh(x)	Arc-hyperbolic cosine
$atanh(x)	Arc-hyperbolic tangent

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

E.6 Expression language formal syntax (BNF)

The formal syntax is described using Backus-Naur Form (BNF). The syntax of the expression language source is derived from the starting symbol `constant_expression` (see E.6.3.3).

The following meta-syntax conventions used are used here.

- Keywords and punctuation are in **red** text.
- Syntactic categories are named in non-bold text.
- A vertical bar (`|`) separates alternatives.
- Square brackets (`[ ]`) enclose optional items.
- Braces (`{ }`) enclose items that can be repeated zero or more times.

E.6.1 Declarations: declaration data types

casting_type ::= simple_type | constant_primary | signing

integer_type ::= integer_vector_type | integer_atom_type

integer_atom_type ::= **byte** | **shortint** | **int** | **longint**

integer_vector_type ::= **bit**

non_integer_type ::= **shortreal** | **real**

signing ::= **signed** | **unsigned**

simple_type ::= integer_type | non_integer_type

E.6.2 Behavioral statements: Case statements

E.6.2.1 Patterns

assignment_pattern ::= **'{** constant_expression { **,** constant_expression } **}**
| **'{** array_pattern_key **:** constant_expression { **,** array_pattern_key **:** constant_expression } **}**
| **'{** constant_expression **{** constant_expression { **,** constant_expression } **} }**

array_pattern_key ::= constant_expression | assignment_pattern_key

assignment_pattern_key ::= simple_type

assignment_pattern_expression ::= [assignment_pattern_expression_type] assignment_pattern

assignment_pattern_expression_type ::= integer_atom_type

constant_assignment_pattern_expression ::= assignment_pattern_expression

E.6.2.2 Covergroup declarations

open_range_list ::= open_value_range { **,** open_value_range }

open_value_range ::= value_range

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

E.6.3 Expressions

E.6.3.1 Concatenations

constant_concatenation ::= **{** constant_expression { **,** constant_expression } **}**

constant_multiple_concatenation ::= **{** constant_expression constant_concatenation **}**

constant_streaming_expression ::= **{** stream_operator [slice_size] constant_stream_concatenation **}**

stream_operator ::= >> | <<

slice_size ::= simple_type | constant_expression

constant_stream_concatenation ::= **{** constant_stream_expression { **,** constant_stream_expression } **}**

constant_stream_expression ::= constant_expression [**with [** constant_array_range_expression **]**]

constant_array_range_expression ::= constant_expression
| constant_expression **:** constant_expression
| constant_expression **+:** constant_expression
| constant_expression **-:** constant_expression

E.6.3.2 Subroutine calls

math_function_call ::= math_function_identifier [~~(~~**(** list_of_arguments ~~)~~**)**]
list_of_arguments ::= [constant_expression] { **,** [constant_expression] }

E.6.3.3 Expressions

constant_expression ::= constant_primary
| unary_operator constant_primary
| constant_expression binary_operator constant_expression
| constant_expression **?** constant_expression **:** constant_expression
| constant_inside_expression

~~constant_mintypmax_expression ::= constant_expression~~
~~| constant_expression : constant_expression : constant_expression~~

constant_range_expression ::= constant_expression | constant_part_select_range

constant_part_select_range ::= constant_range | constant_indexed_range

constant_range ::= constant_expression **:** constant_expression

constant_indexed_range ::= constant_expression **+:** constant_expression
| constant_expression **-:** constant_expression

constant_inside_expression ::= constant_expression **inside** { open_range_list }

value_range ::= constant_expression | **[** constant_expression : constant_expression **]**

E.6.3.4 Primaries

constant_primary ::= primary_literal
| parameter_identifier constant_select
| constant_concatenation [**[** constant_range_expression **]**]
| constant_multiple_concatenation [**[** constant_range_expression **]**]
| **(** constant_~~mintypmax_~~expression **)**
| constant_streaming_expression
| constant_cast
| math_function_call
| constant_assignment_pattern_expression

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

primary_literal ::= number | unbased_unsized_literal | string_literal

constant_cast ::= casting_type **'** **(** constant_expression **)**

constant_bit_select ::= { **[** constant_expression **]** }

constant_select ::= ~~{ **.** member_identifier constant_bit_select } **.** member_identifier }~~ constant_bit_select [**[** constant_part_select_range **]**]

E.6.3.5 Operators

unary_operator ::= **+** | **-** | **!** | ~ | **&** | **~&** | | | **~|** | ^ | **~^** | **^~**

binary_operator ::= **+** | **-** | ***** | / | **%** | **==** | **!=** | **&&** | **||** | ******
| < | <= | > | >= | **&** | | | ^ | ^~ | ~^ | >> | << | >>> | <<< | -> | <->

E.6.3.6 Numbers

See subclause A.8.7 in IEEE Std 1800-2012.

~~number ::= integral_number | real_number~~

~~integral_number ::= decimal_number | octal_number | binary_number | hex_number~~

~~decimal_number ::= unsigned_number~~
~~| [size] decimal_base unsigned_number~~
~~| [size] decimal_base { _ }~~

~~binary_number ::= [size] binary_base binary_value~~

~~octal_number ::= [size] octal_base octal_value~~

~~hex_number ::= [size] hex_base hex_value~~

~~sign ::= + | -~~

~~size ::= non_zero_unsigned_number~~

~~non_zero_unsigned_number[1] ::= non_zero_decimal_digit { _ | decimal_digit }~~

~~real_number ::= fixed_point_number~~
~~| unsigned_number [. unsigned_number] exp [sign] unsigned_number~~

~~fixed_point_number ::= unsigned_number . unsigned_number~~

~~exp ::= e | E~~

~~unsigned_number ::= decimal_digit { _ | decimal_digit }~~

~~binary_value ::= binary_digit { _ | binary_digit }~~
~~octal_value ::= octal_digit { _ | octal_digit }~~

~~hex_value ::= hex_digit { _ | hex_digit }~~

~~decimal_base ::= '[s|S]d | '[s|S]D~~

~~binary_base ::= '[s|S]b | '[s|S]B~~

~~octal_base ::= '[s|S]o | '[s|S]O~~

~~hex_base ::= '[s|S]h | '[s|S]H~~

~~non_zero_decimal_digit ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9~~

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

~~decimal_digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9~~

~~binary_digit ::= 0 | 1~~

~~octal_digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7~~

~~hex_digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f | A | B | C | D | E | F~~

~~unbased_unsized_literal ::= '0 | '1~~

E.6.3.7 Strings

string_literal ::= " { Any_ASCII_Characters } "

E.6.4 General: Identifiers

identifier ::= \i-[:|$]][\c-[:]]*

~~member_identifier ::= identifier~~

parameter_identifier ::= identifier

math_function_identifier ::= integer_math_function_identifier | real_math_function_identifier

integer_math_function_identifier ::= **$clog2**

real_math_function_identifier ::= **$asin** | **$ln** | **$acos** | **$log10** | **$atan** | **$exp** | **$atan2** | **$sqrt** | **$hypot** | **$pow** | **$sinh** | **$floor** | **$cosh** | **$ceil** | **$tanh** | **$sin** | **$asinh** | **$cos** | **$acosh** | **$tan** | **$atanh**

E.7 SystemVerilog conversion steps

To be able to verify the IP-XACT Expression Language using a SystemVerilog parser or tool, the IP-XACT Expression Language needs to be converted to the SystemVerilog Expression language.

This can be achieved using the following (simple) steps.

E.7.1 Convert parameter

a) Convert the vector(s) left and right information to SystemVerilog (packed) array indexes.

b) Normalize the characters in the name of the parameter to valid SystemVerilog characters by replacing all `-`, `:`, and `.` characters and non US-ASCII characters with a predefined character (or set of characters), e.g., use an underscore (_) instead.

c) Escape all identifiers which could be interpreted as SystemVerilog keywords.

d) Convert the array(s) left and right information to SystemVerilog (unpacked) array indexes.

e) Convert the value and array values (see E.7.2).

E.7.2 Convert expression

Convert all identifiers by normalizing the characters: use the same mechanism as specified by steps c and d in E.7.1.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

E.8 SystemVerilog reference

The following sub clauses of IEEE1800-2012 have been used to determine the IP-XACT Expression Language functionality.

6 - Data Types, of which the following sub clauses should be considered.

- **6.1 - General**, only the parts discussing the type operators, compatibility, and casting.
- **6.2 - Data types and data objects**, should be considered, apart from any distinctions noted in E.2.
- **6.3 - Value Set**, only the sub clause about the logical `0` and `1` values (**6.3.1**); the sub clauses about strength should not be taken into consideration.
- **6.4 - Singular and aggregate types**, **6.5 - Nets and variables**, **6.6 - Net types**, **6.7 - Net declarations**, and **6.8 - Variable declarations**, should not be considered.
- **6.9 - Vector declarations**, only bit vectors should be considered. Also the declaration of vectors is handled by an IP-XACT construct and can be ignored.
- **6.10 - Implicit declarations**, can be ignored because identifiers can only be created using the IP-XACT parameter mechanism
- **6.11 - Integer data types**, only **byte**, **shortint**, **int**, **longint**, and **bit** should be considered, these are 2-state data-types which means all 4-state data types can be ignored.
- **6.12 - Real, shortreal and realtime data types**, only the real data-type should be considered.
- **6.13 - Void data type**, **6.14 - Chandle data type**, and **6.15 - Class**, are not supported by the IP-XACT Expression Language and should not be considered.
- **6.16 - String data type**, the IP-XACT string data-type functionality has been simplified considerably; only the equality, inequality, concatenation and replication operators should be considered.
- **6.17 - Event data type**, **6.18 - User-defined types**, **6.19 - Enumerations**, and **6.20 - Constants**, cannot be specified using the IP-XACT expression language and should not be considered.
- **6.21 - Scope and lifetime**, is handled by the *IP-XACT XML Format* which is outside of the IP-XACT Expression Language.
- **6.22 - Type compatibility**, only the parts which discuss the **byte**, **shortint**, **int**, **longint**, **bit**, **shortreal**, and **real** data-types should be considered.
- **6.23 - Type operator**, the type operator is not supported and should not be considered.
- **6.24 - Casting**, only the parts which discuss the **byte**, **shortint**, **int**, **longint**, **bit**, **shortreal**, and **real** data-types should be considered.
- **6.25 - Parameterized data types**, should not be considered.

7 - Aggregate data types, of which only the sub clause about packed arrays and unpacked arrays should be considered.

- **7.1 - General**, only section on packed and unpacked arrays should be considered.
- **7.2 - Structures** and **7.3 – Unions**, are not supported and should not be considered.
- **7.4 - Packed and unpacked arrays,** should be fully considered.
- **7.5 - Dynamic arrays**, are not supported and should not be considered.
- **7.6 - Array assignments**, are handled by the *IP-XACT XML Format*; however, the format of the assignment should still be considered.
- **7.7 - Arrays as arguments to subroutines,** subroutines are not supported and should not be considered.
- **7.8 - Associative arrays**, are not supported and should not be considered.
- **7.9 - Associative array methods**, are not supported and should not be considered.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

- **7.10 - Queues**, are not supported and should not be considered.
- **7.11 - Array querying functions** and **7.12 - Array manipulation methods**, are not supported and should not be considered.

10 – Assignment statements, of which only sub clause **10.9 - Assignment patterns**, should be considered, in particular **10.9.1**, which clarifies the array assignment patterns.

11 - Operators and expressions, of which the following sub clauses should be considered.

- **11.1 - General**, should be considered, apart from the operators and expressions discussed in E.4.
- **11.2 - Overview**, where for the operand sub clause only, constant literal numbers, string literals, packed array, and unpacked arrays should be considered.
- **11.3 - Operators**, only the operators identified for the IP-XACT Expression Language should be considered and only on the data-types allowed. The assignment within an expression section can be ignored.
- **11.4 - Operator descriptions**, only Arithmetic operators, Relational operators, Equality operators (`x` and `z` excluded), Logical operators, Bitwise operators, Reduction operators, Shift operators, Conditional operators, Concatenation, Set membership, and streaming operators should be considered.
- **11.5 - Operands**, should be considered.
- **11.6 - Expression bit lengths**, should be considered for the **byte**, **shortint**, **int**, **longint**, **bit**, **shortreal**, and **real** values.
- **11.7 - Signed expressions**, the `$signed` and `$unsigned` system functions are not supported and should be ignored.
- **11.8 - Expression evaluation rules**, should be considered, however the sub clauses about evaluating an assignment and handling of `X` and `Z` values can be ignored.
- **11.9 - Tagged union expressions and member access**, unions are not supported and can be ignored.
- **11.10 - String literal expressions**, are not supported and can be ignored.
- **11.11 - Operator overloading**, is not supported and can be ignored.
- **11.12 - Minimum, typical, and maximum delay expressions**, delay expressions are not supported and can be ignored.
- **11.13 - Let construct**, this construct is not supported and can be ignored.

20 – Utility system tasks and system functions, of which only sub clause **20.8 – Math functions**, should be considered.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Annex F

(normative)

Tight generator interface

IP-XACT generators are tools that are invoked from within a DE to perform an operation required by the user of the DE. For example, generators can be provided to verify the configuration of a subsystem, generate an address map, or write a netlist representation of the subsystem in a target language such as Verilog or SystemC. To perform their various operations, most generators need access to the IP-XACT meta-data describing the subsystem, as currently loaded into the DE. Generators need both read- and write-access to the IP-XACT meta-data. All generators are external applications running in a separate address space from the DE.

The TGI defines how the DE and generator cooperate to achieve the desired end-goal of the user of the DE. The TGI defines the method of communication between the DE and generator, the method for invoking the generator, and the actual application programming interface (API) that can be used to read and write the IP-XACT meta-data stored in the DE. F.1, F.2, and F.3 describe each of these three aspects of the TGI, respectively.

F.1 Method of communication

The DE and the generator communicate with each other by sending messages to each other utilizing the SOAP standard. SOAP provides a simple means for sending XML-format messages using HTTP or other transport protocols. The TGI restricts the set of allowed transport protocols to HTTP and a file-based protocol. All generators are required to support the HTTP protocol, but support for the file-based protocol is optional. The same rules apply to the DE—it shall support the use of the HTTP protocol, but is not required to support the file-based protocol, even though a generator may allow it. The protocols supported by a generator are specified using the **transportMethod** element within the **componentGenerator** element.

The information required to use a particular transport protocol shall be passed to the generator by the DE when it is invoked, as described in F.2. For the HTTP protocol, the generator is passed a URL of the form **http://***host_name***:***port_number*. All SOAP messages sent to the DE shall be sent using the referenced URL. For the file-based protocol, the generator is passed a URL of the form **file://***file_name*. In this case, all SOAP messages are written to the specified file.

Each DE and generator is responsible for setting itself up to communicate using SOAP with the appropriate transport protocol. For example, a generator written in Tcl might include the Tcl SOAP package to enable SOAP functionality. Once the communication channel is set up, the generator can read and write the IP-XACT meta-data using any legal SOAP message. The set of legal SOAP messages defines the API portion of the TGI (see F.7).

F.2 Generator invocation

All of the information known by the DE about a particular generator comes from an instance of the **componentGenerator** (see 6.13), **abstractorGenerator** (see 8.7), or **generator** (see 9.4) elements. These elements provides the following information.

a) **name** is the name of the generator as seen within the DE.

b) **executable** is the URL defining the location of the generator.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

c) **parameters** is a list of name/value pairs defining information to be passed to the generator.
d) **apiType** indicates the generator type: **TGI** or **none** (no communication).
e) **transportMethods** show any transport mechanisms supported (in addition to HTTP).
f) **phase** (not relevant to the TGI).
g) **vendorExtensions** (not relevant to the TGI).
h) **group** (not relevant to the TGI).

F.2.1 Resolving the URL

The URL defining the generator executable shall resolve to one of the following forms.

— **file:***path_to_executable* (e.g., `file:/usr/jdoe/bin/mygen.pl` or `file:../bin/mygen.pl`) defines the path for invoking the generator on the machine from which the DE was invoked.
— **file://***machine_name***/***path_to_executable* (e.g., `file://server1/tmp/othergen.pl`) defines the path for invoking a generator on the specified machine.
— **http://***web_address***:***port_number* (e.g., `http://www.acme.com/generator:1500`) defines the URL of a generator implemented as a Web-based server.

All *file references* are relative to the location of the XML description in which the file reference is contained.

For the file-based generators, the DE shall invoke the generator as a sub-process with a command line built up as:

executable **-url** *transport_URL generator_parameter_arguments*

The *generator_parameter_arguments* are the parameters from the **componentGenerator** element with the user-specified values. Each parameter causes two additional arguments to be passed to the generator with the following format: **-***parameter_name parameter_value*. The DE needs to create a *transport_URL* that specifies a protocol supported by the generator as defined by the transport methods within the **componentGenerator**. The DE is also responsible for ensuring any passed parameters can be interpreted correctly. This URL is to be used in the generator to set up the SOAP communication channel.

For Web-based generators, the DE shall send a message to the address and port defined as the executable. The format of this message is

url=transport_URL&*generator_parameter_arguments*

In this case, the generator parameters are formatted using the standard HTTP parameter passing syntax. The specified transport URL shall be used by the generator for any return messages to the DE.

The invocation syntax described above applies only to generators with an API type of **TGI**. Generators with an API type of **none** are invoked as described above, excluding the **transport_URL** argument.

F.2.2 Example

This example shows file-based and Web-based **componentGenerator** elements.

```
<ipxact:componentGenerator>
   <ipxact:name>myGenerator</ipxact:name>
   <ipxact:parameters>
      <ipxact:parameter resolve="user">
```


Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
            <ipxact:name>param1</ipxact:name>
            <ipxact:value>default1</ipxact:value>
         </ipxact:parameter>
         <ipxact:parameter>
            <ipxact:name>param2</ipxact:name>
            <ipxact:value>fixedValue</ipxact:value>
         </ipxact:parameter>
      </ipxact:parameters>
      <ipxact:apiType>TGI_2014_BASE</ipxact:apiType>
      <ipxact:transportMethods>
         <ipxact:transportMethod>file</ipxact:transportMethod>
      </ipxact:transportMethods>
      <ipxact:generatorExe>../bin/myGenerator.pl</ipxact:generatorExe>
   </ipxact:componentGenerator>
```

produces the following output.

```
path_to _XML/../bin/myGenerator -url http://host:port -param1 default1
    -param2 fixedValue
```

Whereas:

```
<ipxact:componentGenerator>
   <ipxact:name>myWebGenerator</ipxact:name>
   <ipxact:parameters>
      <ipxact:parameter resolve="user">
         <ipxact:name>param1</ipxact:name>
         <ipxact:value>default1</ipxact:value>
      </ipxact:parameter>
      <ipxact:parameter>
         <ipxact:name>param2</ipxact:name>
         <ipxact:value>fixedValue</ipxact:value>
      </ipxact:parameter>
   </ipxact:parameters>
   <ipxact:apiType>TGI_2014_BASE</ipxact:apiType>
   <ipxact:generatorExe>http://www.acme.com:1500</ipxact:generatorExe>
</ipxact:componentGenerator>
```

produces the following output.

```
http://www.acme.com:1500?url=http%3a%2f%2fhost%3aport&param1=default1
   &param2=fixedValue
```

F.3 TGI API

The TGI API defines the set of legal SOAP messages that can be sent from a generator to a DE, along with the format of the responses the generator can expect from a given request (message) to the DE. The API

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

shall provide the means of getting and setting values within the IP-XACT design currently represented in the DE. The API commands can be classified as shown in Table F.1.

Table F.1—TGI API classifications

Category	Description	Example
Get	Commands which get attribute or element values. These commands are available for getting all information from the design and component schemas. If the attribute or element does not exist, this may return a default value, an empty string, or an empty array.	Get port width.
Set	Commands which set element values. These commands are available to set each element for which the **resolve** attribute is legal. Setting the value of the element fails unless the resolve value is **user** or **generator**. Set routines return a Boolean value where a *true* return code implies a successful operation. If **false** is returned, the SOAP fault code shall provide additional information detailing the failure.	Get parameter value.
Traversal	Commands that return a list of elements, which can then be traversed for further manipulation.	Get components in a design.
Administrative	Commands that do not deal directly with the IP-XACT meta-data.	Terminate communication.

The complete set of API commands is defined using WSDL so that it can be defined in a language-independent format.

F.3.1 TGI fault codes

The fault codes for TGI failures are as follows:

1 - Unknown (undefined) error

2 - Illegal element ID

3 - Illegal value(s)

4 - Element is not modifiable (incompatible **resolve** value)

5 - Operation not supported by the DE

6 - Operation not supported in this version of the schema

7 - Operation failed

F.3.2 Administrative commands

There are three administrative commands defined in the API.

a) **Init** is the required first message from the generator to the DE. It tells the DE that the generator has properly connected via SOAP.

   1) Input

      i) **apiVersion** of type *string*—Indicates the API version for which the generator is defined to work.

      ii) **failureMode** of type ***apiFailureMode***—Compatibility failure mode

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

fail indicates the DE shall return an error on the `init` call if its API version does not match the one passed to the `init` call;

error indicates the DE shall return an error each time a potentially incompatible API call is made;

warning indicates the DE shall increment a warning count each time a potentially incompatible API call is made.

iii) **message** of type *string*—Message that the DE may display to the user.

2) Returns: **status** of type ***boolean***.

b) **End** is the required last message from the generator to the DE. It tells the DE it is okay to stop listening for messages from the generator. This includes a generator return status, although the generator is not strictly required to terminate after sending the message.

c) **Message** indicates some form of generator status to pass to the user.

F.4 IDs and configurable values

The *handles* in TGI are classified as ***instanceID***s and ***ID***s. The ***instanceID***s reference configured entities while the ***ID***s reference unconfigured entities. The following TGI calls show the difference between ***instanceID*** and ***ID*** explicitly for top-level elements and configurable elements.

instanceID = **abstractionDefInstanceID** | **busDefInstanceID** | **componentInstanceID** | **abstractorInstanceID** | **designInstanceID** | **designConfigurationInstanceID**

ID = **abstractionDefID** | **busDefID** | **componentID** | **abstractorID** | **designID** | **designConfigurationID** | **generatorChainID**

For each ***instanceID***, the **getUnconfiguredID**(***instanceID***) can be called to retrieve the ***ID***. For the calls that retrieve information (get calls), it is presumed the call on an ***instanceID*** also returns ***instanceID***s, e.g. **getComponentBusInterfaceIDs**(***componentInstanceID***) returns **busInterfaceInstanceID**s (configured), whereas **getComponentBusInterfaceIDs**(***componentID***) returns **busInterfaceID**s (unconfigured). For clarity, the distinction between***instanceID***s and ***ID***s has not been made for non-top-level elements. Calls that modify information (edit or set calls), operate only on ***ID***s (unconfigured), e.g., **addComponentBusInterface**(***componentID***) returns a new unconfigured **busInterfaceID**, whereas the call **addComponentBusInterface**(***componentInstanceID***) fails.

Handles returned by TGI commands are persistent for the duration of a single generator invocation provided the element being referenced is not removed. For example, if a handle represents an address space element, that handle can be utilized as often as is needed during a single generator invocation, unless the component containing the address map is removed via **removeDesignComponentInstance()**. Furthermore, persistent TGI handles to the same object are identical for the duration of the generator invocation. This enables generators to identify objects by means of their handles.

F.5 TGI messages

The TGI is a set of messages used to query and modify an IP-XACT compliant database. The TGI messages are composed of a SOAP envelope and a TGI body. The TGI services are specified in the `TGI.wsdl` file. Each TGI body message is an XML element whose name is the name of the TGI command and whose elements are the arguments of the TGI command. All TGI messages apply to IP-XACT XML elements, identified by an ID, i.e., a TGI server-defined constant uniquely identifying an IP-XACT XML element throughout a TGI server session.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.6 Vendor attributes

One case of special interest to a user may be the location of vendor attributes in the schema. These attributes are allowed in more places in the schema than the TGI allows a user to retrieve them. This goes back to the concept where one function uses many different ID types to return some data. In the case of vendor attributes, these can only be accessed if the containing element has an ID.

F.7 TGI calls

This subclause details the TGI API calls. F.7.1 is an index of the various categories and F.7.2 is an index for the specific messages within each of those categories. The actual API breakouts start with subclause F.7.3.

F.7.1 Category index

Base category name	Extended category name
(BASE)	(EXTENDED)
Abstraction definition operations (BASE)	Abstraction definition operations (EXTENDED)
Abstractor operations (BASE)	Abstractor operations (EXTENDED)
Abstractor view operations (BASE)	Abstractor view operations (EXTENDED)
Access handle operations (BASE)	Access handle operations (EXTENDED)
Address space operations (BASE)	Address space operations (EXTENDED)
Array operations (BASE)	Array operations (EXTENDED)
Assertion operations (BASE)	
BitsInLau get operations (BASE)	BitsInLau get operations (EXTENDED)
Bridge operations (BASE)	Bridge operations (EXTENDED)
Bus definition operations (BASE)	Bus definition operations (EXTENDED)
Businterface operations (BASE)	Businterface operations (EXTENDED)
	Catalog operations (EXTENDED)
Channel operations (BASE)	Channel operations (EXTENDED)
Choice operations (BASE)	Choice operations (EXTENDED)
Component instantiation operations (BASE)	Component instantiation operations (EXTENDED)
Component operations (BASE)	Component operations (EXTENDED)
Configurable element operations (BASE)	
Constraint set operations (BASE)	Constraint set operations (EXTENDED)
Cpu operations (BASE)	Cpu operations (EXTENDED)

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Base category name	Extended category name
Design configuration instantiation operations (BASE)	Design configuration instantiation operations (EXTENDED)
Design configuration operations (BASE)	Design configuration operations (EXTENDED)
Design instantiation operations (BASE)	
Design operations (BASE)	Design operations (EXTENDED)
Driver operations (BASE)	Driver operations (EXTENDED)
Element attribute operations (BASE)	Element attribute operations (EXTENDED)
Field operations (BASE)	
File builder operations (BASE)	File builder operations (EXTENDED)
File set operations (BASE)	File set operations (EXTENDED)
Generator chain operations (BASE)	Generator chain operations (EXTENDED)
Generator operations (BASE)	Generator operations (EXTENDED)
Global operations (BASE)	
Indirect interface operations (BASE)	Indirect interface operations (EXTENDED)
Memory map operations (BASE)	Memory map operations (EXTENDED)
Module and type parameter operations (BASE)	Module and type parameter operations (EXTENDED)
Parameter operations (BASE)	
Port operations (BASE)	Port operations (EXTENDED)
Present operations (BASE)	
Register file operations (BASE)	Register file operations (EXTENDED)
Register operations (BASE)	Register operations (EXTENDED)
	Remap state operations (EXTENDED)
Slice operations (BASE)	Slice operations (EXTENDED)
Typedef operations (BASE)	Typedef operations (EXTENDED)
Vector operations (BASE)	Vector operations (EXTENDED)
Vendor extensions operations (BASE)	
View operations (BASE)	View operations (EXTENDED)
Whitebox operations (BASE)	Whitebox operations (EXTENDED)
Whitebox ref operations (BASE)	Whitebox ref operations (EXTENDED)
name group operations (BASE)	name group operations (EXTENDED)

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Base category name	Extended category name
top element operations (BASE)	
All ID types	

F.7.2 Message indexes

F.7.2.1 (BASE)

Name	Description
getRegisterFieldBitOffset	Returns the bitOffset in the given regField element
getRegisterFieldResetIDs	Returns the resetIDs in the given regField element
getResetTypeRef	Returns the resetTypeRef for the given reset element
getResetValue	Returns the value for the given reset element
getResetValueExpression	Returns the value expression for the given reset element
getResetMask	Returns the mask for the given reset element
getResetMaskExpression	Returns the mask expression for the given reset element
getRegisterFieldTypeIdentifier	Returns the typeIdentifier for the given regField element
getRegisterFieldBitWidth	Returns the bitWidth for the given regField element
getRegisterFieldVolatility	Returns the volatility or the given regField element
getRegisterFieldAccess	Returns the access or the given regField element
getRegisterFieldEnumeratedValueIDs	Returns the enumreatedValueIDs for the given regField element
getEnumeratedValueUsage	Returns the usage for the given enumeratedValue element
getEnumeratedValueValue	Returns the value for the given enumeratedValue element
getRegisterFieldModifiedWriteValue	Returns the modifiedWriteValue for the given regField element
getRegisterFieldModifiedWriteValueModify	Returns the modify attribute value of modifiedWriteValue for the given regField element
getRegisterFieldWriteValueConstraintWriteAsRead	Returns the writeAsRead for the given regField element
getRegisterFieldWriteValueConstraintUseEnumerated-Values	Returns the useEnumeratedValues for the given regField element
getRegisterFieldWriteValueConstraintMinMax	Returns the min and max for the given regField element
getRegisterFieldReadAction	Returns the readAction for the given regField element
getRegisterFieldReadActionModify	Returns the modify attribute value of readAction for the given regField element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Name	Description
getRegisterFieldTestable	Returns the testAble for the given regField element
getRegisterFieldTestContraint	Returns the testConstraint for the given regField element
getRegisterFieldReserved	Returns the reserved for the given regField element

F.7.2.2 (EXTENDED)

Name	Description
addRegisterFieldReset	Add a reset with the given type and value to the given regField element
removeRegisterFieldReset	Remove the given reset element
setResetMask	Set the mask with the given value for the given reset element
setRegisterFieldID	Set the FieldID attribute to the given value for the given regField element
setRegisterFieldTypeIdentifier	Set the typeIdentifier to the given value for the given regField element
setRegisterFieldVolatility	Set the volatility to the given value for the given regField element
setRegisterFieldAccess	Set the access to the given value for the given regField element
addRegisterFieldEnumeratedValue	Add a enumeratedValue with the given name and value to the given regField element
removeRegisterFieldEnumeratedValue	Remove the given enumeratedValue element
setRegisterFieldEnumeratedValueUsage	Set the usage with the given value for the given enumeratedValue element
setRegisterFieldModifiedWriteValue	Set the modifiedWriteValue with the given type for the given regField element
setRegisterFieldModifiedWriteValueModify	Set the modify attribute of modifiedWriteValue with the given value for the given regField element
setRegisterFieldWriteValueContraint	Set the writeValueContraint elements writeAsRead, useEnumeratedValues, min, and max with the given values for the given regField element
setRegisterFieldReadAction	Set the readAction with the given type for the given regField element
setRegisterFieldReadActionModify	Set the modify attribute of readAction with the given value for the given regField element
setRegisterFieldTestable	Set the testable with the given value for the given regField element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Name	Description
setRegisterFieldTestContraint	Set the testConstraint with the given value for the given regField element
setRegisterFieldReserved	Set the reserved with the given value for the given regField element

F.7.2.3 Abstraction definition operations (BASE)

Name	Description
getAbstractionDefinitionBusType	Returns busType for the given abstractionDef or abstractionDefInstance element
getAbstractionDefinitionExtends	Returns extends for the given abstractionDef or abstractionDefInstance element
getAbstractionDefinitionPortIDs	Returns abstractionDefPortIDs for the given abstractionDef or abstractionDefInstance element
getAbstractionDefPortLogicalName	Returns logicalName for the given abstractionDefPort element
getAbstractionDefPortStyle	Returns portStyle for the given abstractionDefPort element
getAbstractionDefPortIsAddress	Returns isAddress for the given abstractionDefPort element
getAbstractionDefPortIsData	Returns isData for the given abstractionDefPort element
getAbstractionDefPortIsClock	Returns isClock for the given abstractionDefPort element
getAbstractionDefPortIsReset	Returns isReset for the given abstractionDefPort element
getAbstractionDefPortModeIDs	Returns abtractionDefPortModeIDs for the given abstractionDefPort element
getAbstractionDefPortModeGroup	Returns group for the given abstractionDefPortMode element
getAbstractionDefPortModePresence	Returns presence for the given abstractionDefPortMode element
getAbstractionDefPortModeWidth	Returns width for the given abstractionDefPortMode element
getAbstractionDefPortModeDirection	Returns direction for the given abstractionDefPortMode element
getAbstractionDefPortModeConstraintIDs	Returns abstractionDefPortModeConstraintIDs for the given abstractionDefPortMode element
getAbstractionDefPortModeDriveConstraintIDs	Returns driveContraintIDs for the given abstractionDefPortModeConstraint element
getAbstractionDefPortModeLoadConstraintIDs	Returns loadConstraintIDs for the given abstractionDefPortModeConstraint element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Name	Description
getAbstractionDefPortModeTimingConstraintIDs	Returns timingContraintIDs for the given abstractionDefPortModeConstraint element
getAbstractionDefPortMirroredModeConstraintIDs	Returns mirrored abstractionDefPortModeConstraintIDs for the given abstractionDefPortMode element
getAbstractionDefPortDefaultValue	Returns defaultValue for the given abstractionDefPort element
getAbstractionDefPortRequiresDriver	Returns requiresDriver for the given abstractionDefPort element
getAbstractionDefPortRequiresDriverType	Returns requiresDriverType for the given abstractionDefPort element
getAbstractionDefPortModeInitiative	Returns initiative for the given abstractionDefPortMode element
getAbstractionDefPortModeKind	Returns kind for the given abstractionDefPortMode element
getAbstractionDefPortModeKindCustom	Returns kind custom for the given abstractionDefPortMode element
getAbstractionDefPortModeBusWidth	Returns busWidth for the given abstractionDefPortMode element
getAbstractionDefPortModeProtocolType	Returns protocolType for the given abstractionDefPortMode element
getAbstractionDefPortModeProtocolTypeCustom	Returns protocolType custom for the given abstractionDefPortMode element
getAbstractionDefPortModeProtocolPayloadName	Returns payloadName for the given abstractionDefPortMode element
getAbstractionDefPortModeProtocolPayloadType	Returns payloadType for the given abstractionDefPortMode element
getAbstractionDefPortModeProtocolPayloadExtension	Returns payloadExtension for the given abstractionDefPortMode element
getAbstractionDefPortModeProtocolPayloadExtensionMandatory	Returns payloadExtension mandatory for the given abstractionDefPortMode element

F.7.2.4 Abstraction definition operations (EXTENDED)

Name	Description
createAbstractionDefinition	Create a new abstractionDef with the given VLNV, busDefVLNV, logicalName and type and returns its abstractionDefID; fails and returns null if VLNV already exists
setAbstractionDefinitionExtends	Set extends with the given value for the given abstractionDef element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Name	Description
addAbstractionDefinitionPort	Add abstractionDefPort with the given logicalName and type to the given abstractionDef element
removeAbstractionDefinitionPort	Remove the given abstractionDefPort element
setAbstractionDefPortAddress	Set isAddress with the given value for the given abstractionDefPort element
setAbstractionDefPortData	Set isData with the given value for the given abstractionDefPort element
setAbstractionDefPortClock	Set isClock with the given value for the given abstractionDefPort element
setAbstractionDefPortReset	Set isReset with the given value for the given abstractionDefPort element
addAbstractionDefPortMode	Add abstractionDefPortMode with the given value for the given abstractionDefPort element
setAbstractionDefPortModeGroup	Set group with the given value for the given abstractionDefPortMode element
setAbstractionDefPortModePresence	Set presence with the given value for the given abstractionDefPortMode element
setAbstractionDefPortModeWidth	Set width with the given value for the given abstractionDefPortMode element
setAbstractionDefPortModeDirection	Set direction with the given value for the given abstractionDefPortMode element
addAbstractionDefPortModeDriveConstraint	Add driveContraint with the given type for the given abstractionDefPortMode element
addAbstractionDefPortModeLoadConstraint	Add loadContraint with the given type for the given abstractionDefPortMode element
addAbstractionDefPortModeTimingConstraint	Add timingContraint with the given type for the given abstractionDefPortMode element
addAbstractionDefPortMirroredModeDriveConstraint	Add mirrored driveContraint with the given type for the given abstractionDefPortMode element
addAbstractionDefPortMirroredModeLoadConstraint	Add mirrored loadContraint with the given type for the given abstractionDefPortMode element
addAbstractionDefPortMirroredModeTimingConstraint	Add mirrored timingContraint with the given type for the given abstractionDefPortMode element
setAbstractionDefPortDefaultValue	Set defaultValue with the given value for the given abstractionDefPort element
setAbstractionDefPortRequiresDriver	Set requiresDriver with the given value for the given abstractionDefPort element
setAbstractionDefPortRequiresDriverType	Set driverType attribute of requiresDriver with the given value for the given abstractionDefPort element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Name	Description
setAbstractionDefPortModeInitiative	Set initiative with the given value for the given abstractionDefPort element
setAbstractionDefPortModeKind	Set kind with the given value for the given abstractionDefPort element
setAbstractionDefPortModeKindCustom	Set custom attribute of kind with the given value for the given abstractionDefPort element
setAbstractionDefPortModeBusWidth	Set busWidth with the given value for the given abstractionDefPort element
setAbstractionDefPortModeProtocolType	Set protocolType with the given value for the given abstractionDefPort element
setAbstractionDefPortModeProtocolTypeCustom	Set custom attribute of protocolType with the given value for the given abstractionDefPort element
setAbstractionDefPortModeProtocolPayloadName	Set payloadName with the given value for the given abstractionDefPort element
setAbstractionDefPortModeProtocolPayloadType	Set payloadType with the given value for the given abstractionDefPort element
setAbstractionDefPortModeProtocolPayloadExtension	Set payloadExtension with the given value for the given abstractionDefPort element
setAbstractionDefPortModeProtocolPayloadExtension-Mandatory	Set mandatory attribute of payloadExtension with the given value for the given abstractionDefPort element

F.7.2.5 Abstractor operations (BASE)

Name	Description
getAbstractorAbstractorMode	Returns the abstractorMode for the given abstractor or abstractorInstance element
getAbstractorBusType	Returns the busType for the given abstractor or abstractorInstance element
getAbstractorAbstractorInterfaceIDs	Returns the abstractorInterfaceIDs for the given abstractor or abstractorInstance element
getAbstractorBusInterfaceAbstractionTypeIDs	Returns the abstractionTypeIDs for the given abstractorBusInterface element
getAbstractorViewIDs	Returns the abstractorViewIDs for the given abstractor or abstractorInstance element
getAbstractorComponentInstantiationsID	Returns the componentInstantationIDs for the given abstractor or abstractorInstance element
getAbstractorPortIDs	Returns the portIDs for the given abstractor or abstractorInstance element
getAbstractorGeneratorIDs	Returns the generatorIDs for the given abstractor or abstractorInstance element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Name	Description
getAbstractorChoiceIDs	Returns the choiceIDs for the given abstractor or abstractorInstance element
getAbstractorFileSetIDs	Returns the fileSetIDs for the given abstractor or abstractorInstance element

F.7.2.6 Abstractor operations (EXTENDED)

Name	Description
createAbstractor	Create a new abstractor with the given VLNV and returns its abstractorID; fails and returns null if VLNV already exists
addAbstractorBusInterfaceAbstractionType	Add an abstractionType with the given abstractionRef for the given abstractorBusInterface element
removeAbstractorBusInterfaceAbstractionType	Remove the given abstractionType element
addAbstractorView	Add an abstractorView with the given name and envIdentifier for the given abstractor element
removeAbstractorView	Remove the given abstractorView element
addAbstractorComponentInstantiation	Add a componentInstantiation with the given name for the given abstractor element
removeAbstractorComponentInstantiation	Remove the given componentInstantiation element
addAbstractorPort	Add a port with the given name, type, and directionOrInitiative for the given abstractor element
removeAbstractorPort	Remove the given port element
addAbstractorGenerator	Add a generator with the given name and path to the given abstractor element
removeAbstractorGenerator	Remove the given generator element
addAbstractorChoice	Add a choice with the given name and enumerations to the given abstractor element
removeAbstractorChoice	Remove the given choice element
addAbstractorFileSet	Add a fileSet with the given name to the given abstractor element
removeAbstractorFileSet	Remove the given fileSet element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.2.7 Abstractor view operations (BASE)

Name	Description
getAbstractorViewEnvIdentifier	Returns envIdentifier for the given abstractorView element
getAbstractorViewComponentInstantiationRef	Returns componentInstantiationRef for the given abstractorView element
getAbstractorViewComponentInstantiationRefID	Returns componentInstantiationID for the given abstractorView element

F.7.2.8 Abstractor view operations (EXTENDED)

Name	Description
setAbstractorViewComponentInstantiationRef	Set componentInstantiationRef for the given abstractorView element

F.7.2.9 Access handle operations (BASE)

Name	Description
getAccessHandleForce	Returns the value of attribute force for a given accessHandle element
getAccessHandleViewIDs	Returns the viewIDs for a given accessHandle element
getAccessHandleIndices	Returns the indices for a given accessHandle element
getAccessHandleSliceIDs	Returns the slideIDs for a given accessHandle element

F.7.2.10 Access handle operations (EXTENDED)

Name	Description
setAccessHandleForce	Set the given value for the attribute force for the given accessHandle element
addAccessHandleViewName	Add a viewRef with the given name to the given accessHandle element
removeAccessHandleViewName	Remove the viewRef with the given name from the given accessHandle element
addAccessHandleSlice	Add a slice with the given pathSegmentNames and the given indices to the given accessHandle element
removeAccessHandleSlice	Remove the given slice

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.2.11 Address space operations (BASE)

Name	Description
getAddressSpaceRange	Returns the range of the given addressSpace element
getAddressSpaceWidth	Returns the width of the given addressSpace element
getAddressSpaceSegmentIDs	Returns the segmentIDs of the given addressSpace element
getSegmentAddressOffset	Returns the addressOffset of the given segment element
getSegmentRange	Returns the range of the given segment element
getAddressSpaceAddressUnitBits	Returns the addressUnitBits for the given addressSpace element
getAddressSpaceExecutableImageIDs	Returns the executableImageIDs for the given addressSpace element
getExecutableImageFileBuilderIDs	Returns the fileBuilderIds for the given executableImage element
getFileBuilderFileType	Returns the fileType for the given fileBuilder element
getFileBuilderCommand	Returns the command for the given fileBuilder element
getFileBuilderFlags	Returns the flags for the given fileBuilder element
getFileBuilderReplaceDefaultFlags	Returns the replaceDefaultFlags for the given fileBuilder element
getExecutableImageLinker	Returns the linker for the given executableImage element
getExecutableImageLinkerFlags	Returns the linkerFlags for the given executableImage element
getExecutableImageLinkerCommandFileID	Returns the linkerCommandFileID for the given executableImage element
getLinkerCommandFileName	Returns the name for a given linkerCommandFile
getLinkerCommandFileLineSwitch	Returns the commandLineSwitch for a given linkerCommandFile
getLinkerCommandFileEnable	Returns the enable for a given linkerCommandFile
getLinkerCommandGeneratorIDs	Returns the generatorIDs for a given linkerCommandFile
getAddressSpaceLocalMemoryMapID	Returns the localMemoryMapID for a given addressSpace element
getLocalMemoryMapAddressBlockIDs	Returns the localAddressBlockIDs for a given localMemory map element
getLocalMemoryMapBankIDs	Returns the localBankIDs for a given localMemory map element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.2.12 Address space operations (EXTENDED)

Name	Description
addAddressSpaceSegment	Add a segment with the given name, addressOffset, and range to the given addressSpace element
removeAddressSpaceSegment	Remove the given segment element
setAddressSpaceAddressUnitBits	Set the addressUnitBits with the given value to the given address space element
addAddressSpaceExecutableImage	Add an executableImage with the given name to the given address space element
removeAddressSpaceExecutableImage	Remove the given executableImage element
addExecutableImageFileBuilderID	Add a fileBuilder with the given fileType and command to the given executableImage element
removeExecutableImageFileBuilderID	Remove the given fileBuilder element
setFileBuilderFlags	Set the flags with the given value for the given fileBuilder element
setFileBuilderReplaceDefaultFlags	Set the replaceDefaultFlags with the given value for the given fileBuilder element
setExecutableImageLinker	Set the linker with the given value for the given executableImage element
setExecutableImageLinkerFlags	Set the linkerFlags with the given value for the given executableImage element
addExecutableImageLinkerCommandFile	Set the linkerCommandFile with the given value for the given executableImage element
removeExecutableImageLinkerCommandFile	Remove the given linkerCommandFile element
addLinkerCommandFileGenerator	Set the generatorRef with the given value for the given linkerCommandFile element
removeLinkerCommandFileGenerator	Remove the given generator element
addLocalMemoryMapAddressBlock	Add an addressBlock with the given name, baseAddress, range, and width to the given localMemoryMap element
removeLocalMemoryMapAddressBlock	Remove the given addressBlock element
addLocalMemoryMapBank	Add a localBank with the given name and baseAddress to the given localMemoryMap element
removeLocalMemoryMapBank	Remove the given localBank element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.2.13 Array operations (BASE)

Name	Description
getArrayIDs	Returns arraryIDs of the given element
getArrayRange	Returns range of the given array element
getArraryRangeExpression	Returns the range expressions of the given array element

F.7.2.14 Array operations (EXTENDED)

Name	Description
addArray	Add an array to the given element
removeArray	Remove the given array

F.7.2.15 Assertion operations (BASE)

Name	Description
getAssertionIDs	Returns assertionIDs of the given element
getAssertionValue	Returns value of the given assertion element
getAssertionValueExpression	Returns expression of the given assertion element
addAssertion	Add an assertion with the given name and given value to the given element
removeAssertion	Remove the given assertion
setAssertionValue	Set the value of the given assertion

F.7.2.16 BitsInLau get operations (BASE)

Name	Description
getBitsInLauValue	Returns bitsInLau value in the given element
getBitsInLauValueExpression	Returns bitsInLau expression in the given element

F.7.2.17 BitsInLau get operations (EXTENDED)

Name	Description
addBitsInLauValue	Add the given bitsInLau expression to the given element
removeBitsInLauValue	Remove the bitsInLau element from the given element
setBitsInLauValue	Set the given bitsInLau expression in the given element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.2.18 Bridge operations (BASE)

Name	Description
getBridgeMasterID	Returns the busInterfaceID of the masterRef attribute in the given bridge element

F.7.2.19 Bridge operations (EXTENDED)

Name	Description
setBridgeMasterID	Set the value of the masterRef attribute for the given busInterfaceID in the given bridge element

F.7.2.20 Bus definition operations (BASE)

Name	Description
getBusDefinitionDirectConnection	Returns directConnection for a given busDef or busDefInstance element
getBusDefinitionBroadcast	Returns broadcast for a given busDef or busDefInstance element
getBusDefinitionIsAddressable	Returns isAddressable for a given busDef or busDefInstance element
getBusDefinitionExtends	Returns extends for a given busDef or busDefInstance element
getBusDefinitionMaxMasters	Returns maxMasters for a given busDef or busDefInstance element
getBusDefinitionMaxSlaves	Returns maxSlaves for a given busDef or busDefInstance element
getBusDefinitionSystemGroupNames	Returns systemGroupNames for a given busDef or busDefInstance element

F.7.2.21 Bus definition operations (EXTENDED)

Name	Description
createBusDefinition	Create a new busDef with the given VLNV, directConnection, and isAddressable and returns its busDefID; fails and returns null if VLNV already exists
setBusDefinitionDirectConnection	Set directionConnection with the given value for the given busDef element
setBusDefinitionBroadcast	Set broadcast with the given value for the given busDef element
setBusDefinitionIsAddressable	Set isAddressable with the given value for the given busDef element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Name	Description
setBusDefinitionExtends	Set extends with the given value for the given busDef element
setBusDefinitionMaxMasters	Set maxMasters with the given value for the given busDef element
setBusDefinitionMaxSlaves	Set maxSlaves with the given value for the given busDef element
setBusDefinitionSystemGroupNames	Set systemGroupNames with the given value for the given busDef element

F.7.2.22 Businterface operations (BASE)

Name	Description
getBusInterfaceBusInstanceID	Returns the busDefInstanceID for a given busInterface element
getBusInterfaceAbstractionTypeIDs	Returns the abstractionTypeIDs for a given busInterface element
getAbstractionTypeViewRefs	Returns the viewRefs for a given abstractionType element
getAbstractionTypeAbstractionInstanceID	Returns the abstractionDefInstanceID for a given abstractionType element
getAbstractionTypePortMapIDs	Returns the portMapIDs for a given abstractionType element
getPortMapLogicalPortID	Returns the logicalPortID for a given portMap element
getPortMapPhysicalPortID	Returns the physicalPortID for a given portMap element
getPortMapRange	Returns the range for a given logicalPort or physicalPort element
getPortMapIndices	Returns the indices for a given physicalPort element
getPortMapLogicalTieOff	Returns the logicalTieOff for a given portMap element
getPortMapIsInformative	Returns the isInformative for a given portMap element
getBusInterfaceInterfaceMode	Returns the interfaceMode for a given busInterface element
getBusInterfaceMasterAddressSpaceID	Returns the addressSpaceID for a given busInterface element (master only)
getBusInterfaceMasterBaseAddress	Returns the baseAddress for a given busInterface element (master only)
getBusInterfaceSlaveMemoryMapID	Returns the memoryMapID for a given busInterface element (slave only)

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Name	Description
getBusInterfaceSlaveBridgeIDs	Returns the bridgeIDs for a given busInterface element (slave only)
getBusInterfaceSlaveFileSetGroupIDs	Returns the fileSetGroupIDs for a given busInterface element (slave only)
getFileSetGroupName	Returns the name for a given fileSetGroup element
getFileSetGroupFileSetIDs	Returns the fileSetIDs for a given fileSetGroup element
getBusInterfaceGroupName	Returns the groupName for a given busInterface element (system, mirroredSystem, and group only)
getBusInterfaceMirroredSlaveRemapAddressIDs	Returns the remapAddressIDs for a given busInterface element (mirroredSlave only)
getBusInterfaceMirroredSlaveRange	Returns the ranges for a given busInterface element (mirroredSlave only)
getBusInterfaceMonitorInterfaceMode	Returns the monitor interfaceMode attribute value for a given busInterface element (monitor only)
getBusInterfaceConnectionRequired	Returns the connectionRequired for a given busInterface element
getBusInterfaceBitSteering	Returns the bitSteering for a given busInterface element
getBusInterfaceEndianness	Returns the endianness for a given busInterface element

F.7.2.23 Businterface operations (EXTENDED)

Name	Description
addBusInterfaceAbstractionType	Add an abstractionType with the given abstractionRef to the given busInterface element
removeBusInterfaceAbstractionType	Remove the given abstractionType element
addAbstractionTypeViewRef	Add a viewRef with the given name to the given abstractionType element
removeAbstractionTypeViewRef	Remove the viewRef element with the given viewName
addAbstractionTypePortMap	Add a portMap with the given name, logicalPortName, and physicalPortNameOrTieValue to the given abstractionType
removeAbstractionTypePortMap	Remove the given portMap element
setPortMapRange	Set the range of the given logicalPort or physicalPort element
setPortMapIndices	Set the indices for the given physicalPort element
setPortMapIsInformative	Set the isInformative for the given portMap element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Name	Description
setBusInterfaceMasterAddressSpaceName	Set the addressSpaceRef with the given name for the given busInterface element (master only)
setBusInterfaceMasterBaseAddress	Set the baseAddress with the given expression for the given busInterface element (master only)
setBusInterfaceSlaveMemoryMapName	Set the memoryMapRef with the given name for the given busInterface element (slave only)
addBusInterfaceSlaveBridge	Add a transparentBridge with the given name for the given busInterface (slave only)
removeBusInterfaceSlaveBridge	Remove the given bridge element
addBusInterfaceSlaveFileSetGroup	Add a fileSetGroup with the given name and fileSetRefs to the given busInterface element (slave only)
removeBusInterfaceSlaveFileSetGroup	Remove the given fileSetGroup
setBusInterfaceGroupName	Set the group with the given name for the given busInterface element (system, mirroredSystem, and group only)
setBusInterfaceConnectionRequired	Set connectionRequired with the given value for the given busInterface element
setBusInterfaceBitSteering	Set bitSteering with the given value for the given busInterface element
setBusInterfaceEndianness	Set endianness with the given value for the given busInterface element

F.7.2.24 Catalog operations (EXTENDED)

Name	Description
getCatalogCatalogsIpxactFiles	Returns catalogs IpxactFileIDs for the given catalog element
getIpxactFileVLNV	Returns VLNV for the given ipxactFile element
getIpxactFileName	Returns name for the given ipxactFile element
getCatalogBusDefinitionsIpxactFiles	Returns busDefintions IpxactFileIDs for the given catalog element
getCatalogAbstractionDefinitionsIpxactFiles	Returns abstractionDefintions IpxactFileIDs for the given catalog element
getCatalogComponentsIpxactFiles	Returns components IpxactFileIDs for the given catalog element
getCatalogAbstractorsIpxactFiles	Returns abstractors IpxactFileIDs for the given catalog element
getCatalogDesignsIpxactFiles	Returns designs IpxactFileIDs for the given catalog element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Name	Description
getCatalogDesignConfigurationsIpxactFiles	Returns designConfigurations IpxactFileIDs for the given catalog element
getCatalogGeneratorChainsIpxactFiles	Returns generatorChains IpxactFileIDs for the given catalog element
createCatalog	Create a new catalog and returns its catalogID; fails and returns null if VLNV already exists
addCatalogCatalogsIpxactFiles	Add ipxactFile with the given VLNV and name to catalogs in the given catalog element
removeCatalogCatalogsIpxactFiles	Remove the given ipxactFile element
addCatalogBusDefinitionsIpxactFiles	Add ipxactFile with the given VLNV and name to busDefinitions in the given catalog element
removeCatalogBusDefinitionsIpxactFiles	Remove the given ipxactFile element
addCatalogAbstractionDefinitionsIpxactFiles	Add ipxactFile with the given VLNV and name to abstractionDefinitions in the given catalog element
removeCatalogAbstractionDefinitionsIpxactFiles	Remove the given ipxactFile element
addCatalogComponentsIpxactFiles	Add ipxactFile with the given VLNV and name to components in the given catalog element
removeCatalogComponentsIpxactFiles	Remove the given ipxactFile element
addCatalogAbstractorsIpxactFiles	Add ipxactFile with the given VLNV and name to abstractors in the given catalog element
removeCatalogAbstractorsIpxactFiles	Remove the given ipxactFile element
addCatalogDesignsIpxactFiles	Add ipxactFile with the given VLNV and name to designs in the given catalog element
removeCatalogDesignsIpxactFiles	Remove the given ipxactFile element
addCatalogDesignConfigurationsIpxactFiles	Add ipxactFile with the given VLNV and name to designConfigurations in the given catalog element
removeCatalogDesignConfigurationsIpxactFiles	Remove the given ipxactFile element
addCatalogGeneratorChainsIpxactFiles	Add ipxactFile with the given VLNV and name to generatorChains in the given catalog element
removeCatalogGeneratorChainsIpxactFiles	Remove the given ipxactFile element

F.7.2.25 Channel operations (BASE)

Name	Description
getChannelBusInterfaceIDs	Returns the busInterfaceIDs for the given channel element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.2.26 Channel operations (EXTENDED)

Name	Description
addChannelBusInterface	Add a busInterfaceRef with the given name to the given channel element
removeChannelBusInterface	Remove the busInterfaceRef with the given name from the given channel element

F.7.2.27 Choice operations (BASE)

Name	Description
getChoiceEnumerationIDs	Returns choiceEnumerationIDs for the given choice element
getEnumerationText	Returns text for the given choice element
getEnumerationHelp	Returns help for the given choice element

F.7.2.28 Choice operations (EXTENDED)

Name	Description
addChoiceEnumeration	Add enumeration with the given name to the given choice element
removeChoiceEnumeration	Remove the given choiceEnumeration
setEnumerationText	Set text with the given value for the given choiceEnumeration element
setEnumerationHelp	Set help with the given value for the given choiceEnumeration element

F.7.2.29 Component instantiation operations (BASE)

Name	Description
getComponentInstantiationIsVirtual	Returns isVirtual for the given componentInstantiation element
getComponentInstantiationLanguage	Returns language for the given componentInstantiation element
getComponentInstantiationLibraryName	Returns libraryName for the given componentInstantiation element
getComponentInstantiationPackageName	Returns packageName for the given componentInstantiation element
getComponentInstantiationModuleName	Returns moduleName for the given componentInstantiation element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Name	Description
getComponentInstantiationArchitectureName	Returns architectureName for the given componentInstantiation element
getComponentInstantiationConfigurationName	Returns configurationName for the given componentInstantiation element
getComponentInstantiationDefaultFileBuilderIDs	Returns defaultFileBuilderIDs for the given componentInstantiation element
getComponentInstantiationFileSetRefIDs	Returns fileSetRefIDs for the given componentInstantiation element
getFileSetRefLocalName	Returns localName for the given fileSetRef element
getComponentInstantiationContraintSetRefIDs	Returns constraintSetRefIDs for the given componentInstantiation element
getConstraintSetRefLocalName	Returns locaName for the given constraintSetRefRef element
getComponentInstantiationWhiteboxElementRefIDs	Returns whiteboxElementRefIDs for the given componentInstantiation element

F.7.2.30 Component instantiation operations (EXTENDED)

Name	Description
setComponentInstantiationIsVirtual	Set isVirtual with the given value for the given componentInstantiation element
setComponentInstantiationLanguage	Set language with the given value for the given componentInstantiation element
setComponentInstantiationLibraryName	Set libraryName with the given value for the given componentInstantiation element
setComponentInstantiationPackageName	Set packageName with the given value for the given componentInstantiation element
setComponentInstantiationModuleName	Set moduleName with the given value for the given componentInstantiation element
setComponentInstantiationArchitectureName	Set architectureName with the given value for the given componentInstantiation element
setComponentInstantiationConfigurationName	Set configurationName with the given value for the given componentInstantiation element
addComponentInstantiationDefaultFileBuilder	Add defaultFileBuilder with the given fileType to the given componentInstantiation element
removeComponentInstantiationDefaultFileBuilder	Remove the given defaultFileBuilder element
addComponentInstantiationFileSetRef	Add fileSetRef with the given localName to the given componentInstantiation element
removeComponentInstantiationFileSetRef	Remove the given fileSetRef element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Name	Description
addComponentInstantiationConstraintSetRef	Add constraintRefSet with the given localName to the given componentInstantiation element
removeComponentInstantiationConstraintSetRef	Remove the given constraintSetRef element
addComponentInstantiationWhiteboxElementRef	Add whiteboxElementRef with the given name, pathSegmentName, and indices to the given componentInstantiation element
removeComponentInstantiationWhiteboxElementRef	Remove the given whiteboxElementRef element

F.7.2.31 Component operations (BASE)

Name	Description
getComponentBusInterfaceIDs	Returns the busInterfaceIDs for a given component or component instance element
getComponentIndirectInterfaceIDs	Returns the indirectInterfaceIDs for a given component or component instance element
getComponentChannelIDs	Returns the channelIDs for a given component or component instance element
getComponentRemapStateIDs	Returns the remapStateIDs for a given component or component instance element
getComponentAddressSpaceIDs	Returns the addressSpaceIDs for a given component or component instance element
getComponentMemoryMapIDs	Returns the memoryMapIDs for a given component or component instance element
getComponentViewIDs	Returns the viewIDs for a given component or component instance element
getComponentSelectedViewIDs	Returns the selected viewIDs for a given component or component instance element
getComponentComponentInstantiationIDs	Returns the componentInstantiationIDs for a given component or component instance element
getComponentDesignInstantiationIDs	Returns the designInstantiationIDs for a given component or component instance element
getComponentDesignConfigurationInstantiationIDs	Returns the designConfigurationInstantiationIDs for a given component or component instance element
getComponentPortIDs	Returns the portIDs for a given component or component instance element
getComponentGeneratorIDs	Returns the generatorIDs for a given component or component instance element
getComponentChoiceIDs	Returns the choiceIDs for a given component or component instance element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Name	Description
getComponentFileSetIDs	Returns the fileSetIDs for a given component or component instance element
getComponentWhiteboxElementIDs	Returns the whiteboxElementIDs for a given component or component instance element
getComponentCpuIDs	Returns the cpuIDs for a given component or component instance element
getComponentOtherClockDriverIDs	Returns the clockDriverIDs for a given component or component instance element
getComponentResetTypeIDs	Returns the resetTypeIDs for a given component or component instance element

F.7.2.32 Component operations (EXTENDED)

Name	Description
createComponent	Create a new component with the given VLNV and returns its componentID; fails and returns null if VLNV already exists
addComponentBusInterface	Add a busInterface with the given name, busType, interfaceMode, and group to the given component element
removeComponentBusInterface	Remove the given busInterface element
addComponentIndirectInterface	Add an indirectInterface with the given name, indirectAddress, indirectData, memoryMapRef or busInterfaceRef to the given component element
removeComponentIndirectInterface	Remove the given indirectInterface element
addComponentChannel	Add a channel with the given name and busInterfaceRefs to the given component element
removeComponentChannel	Remove the given channel element
addComponentRemapState	Add a remapState with the given name to the given component
removeComponentRemapState	Remove the given remapState element
addComponentAddressSpace	Add an addressSpace with the given name, range, and width to the given component element
removeComponentAddressSpace	Remove the given addressSpace element
addComponentMemoryMap	Add a memoryMap with the given name to the given component element
removeComponentMemoryMap	Remove the given memoryMap element
addComponentView	Add a view with the given name and envIdentifier to the given component element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Name	Description
removeComponentView	Remove the given view element
addComponentComponentInstantiation	Add a componentInstantiation with the given name to the given component element
removeComponentComponentInstantiation	Remove the given componentInstantiation element
addComponentDesignInstantiation	Add a designInstantiation with the given name and designRef to the given component element
removeComponentDesignInstantiation	Remove the given designInstantiation element
addComponentDesignConfigurationInstantiation	Add a designConfigurationInstantiation with the given name and designConfigurationRef to the given component element
removeComponentDesignConfigurationInstantiation	Remove the given designConfigurationInstantiation element
addComponentPort	Add a port with the given name, type, and directionOrInitiative to the given component element
removeComponentPort	Remove the given port element
addComponentGenerator	Add a generator with the given name and path to the given component element
removeComponentGenerator	Remove the given generator element
addComponentChoice	Add a choice with the given name and enumerations to the given component element
removeComponentChoice	Remove the given choice element
addComponentFileSet	Add a fileSet with the given name to the given component element
removeComponentFileSet	Remove the given fileSet element
addComponentWhiteboxElement	Add a whiteboxElement with the given name and type to the given component element
removeComponentWhiteboxElement	Remove the given whiteboxElement element
addComponentCpu	Add a cpu with the given name to the given component element
removeComponentCpu	Remove the given cpu element
addComponentOtherClockDriver	Add a clockDriver with the given name, period, offset, value, and duration to the given component element
removeComponentOtherClockDriver	Remove the given clockDriver element
addComponentResetType	Add a resetType with the given name to the given component element
removeComponentResetType	Remove the given resetType element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.2.33 Configurable element operations (BASE)

Name	Description
getConfigurableElementIDs	Returns all configurableElementIDs of the given unconfigured element
getConfigurableElementValueExpresssion	Returns the value of a given configurable element (default value)
getConfigurableElementValueExpresssion	Returns the expression of a given configurable element (default expression)
getConfigurableElementValueIDs	Returns all configurableElementValueIDs of the given configured element
getConfigurableElementValueValue	Returns the value of a given configurable element value (actual value)
getConfigurableElementValueValueExpresssion	Returns the expression of a given configurable element value (actual expression)
getConfigurableElementValueReferenceID	Returns the referenceID of a given configurable element value
getUnconfiguredID	Returns the unconfiguredID of a given configured element
setConfigurableElementValue	Set the value of a given configurable element (default value)
addConfigurableElementValue	Add a configurable element value with the given expression to the given configured element and the given referenceID
removeConfigurableElementValue	Remove the given configurable element value
setConfigurableElementValueValue	Set the given expression as value for the given configurable element value
setConfigurableElementValueReferenceID	Set the given referenceID for the given configurable element value

F.7.2.34 Constraint set operations (BASE)

Name	Description
getConstraintSetReferenceName	Returns the value of the constraintSetId attribute for the given constraintSet
getConstraintSetRange	Returns the range for the given constraintSet
getConstraintSetDriveConstraints	Returns the driveConstraintIDs for a given constraintSet
getDriveConstraintType	Returns the type for a given driveConstraint element: function class
getDriveConstraintValue	Returns the value for a given driveConstraint element Format depends on the constraint type

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Name	Description
getConstraintSetLoadConstraints	Returns the loadConstraintIDs for a given constraintSet
getLoadContraintType	Returns the type for a given loadConstraint element: function class
getLoadContraintValue	Returns the value for a given loadConstraint element Format is cell function and strength or cell class and strength
getLoadContraintCount	Returns the count for a given loadConstraint element
getConstraintSetTimingConstraints	Returns the timingConstraintIDs for a given constraintSet
getTimingConstraintClockDetails	Returns the clockDetails for a given timingConstraint indicating the clock name, clock edge, and delay type
getTimingConstraintValue	Returns the value for a given timingConstraint element indicating the cycle time percentage

F.7.2.35 Constraint set operations (EXTENDED)

Name	Description
setConstraintSetReferenceName	Set the given referenceName for the constraintSetId attribute for the given constraint set
setConstraintSetRange	Set the given range for the given constraintSet
addConstraintSetDriveConstraint	Add a new driveConstraint with the given type to the given constraintSet
removeConstraintSetDriveConstraint	Remove the given driveConstraint
setDriveContraintValue	Set the given value for the given driveConstraint element
addConstraintSetLoadConstraint	Add a new loadConstraint with the given type to the given constraintSet
removeConstraintSetLoadConstraint	Remove the given loadConstraint
setLoadContraintValue	Set the given value for the given loadConstraint element
setLoadContraintCount	Set the given count for the given loadConstraint element
addConstraintSetTimingConstraint	Add a new timingConstraint with the given clockName to the given constraintSet
removeConstraintSetTimingConstraint	Remove the given timingConstraint
setTimingConstraintValue	Set the given value for the given timingConstraint
setTimingConstraintClockEdge	Set the given clockEdge for the given timingConstraint
setTimingConstraintDelayType	Set the given delayType for the given timingConstraint

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.2.36 Cpu operations (BASE)

Name	Description
getCpuAddressSpaceIDs	Returns addressSpaceIDs for the given cpu element

F.7.2.37 Cpu operations (EXTENDED)

Name	Description
addCpuAddressSpace	Add the given addressSpace name to the given cpu element

F.7.2.38 Design configuration instantiation operations (BASE)

Name	Description
getDesignConfigurationInstantiationLanguage	Returns language for the given designConfigurationInstantiation element
getDesignConfigurationInstantiationDesignConfigurationInstanceID	Returns a designConfigurationInstanceID for the given designConfigurationInstantiation element

F.7.2.39 Design configuration instantiation operations (EXTENDED)

Name	Description
setDesignConfigurationInstantiationLanguage	Set language with the given value for the given designConfigurationInstantiation element

F.7.2.40 Design configuration operations (BASE)

Name	Description
getDesignConfigurationDesignRef	Returns designVLNV for the given designConfiguration element
getDesignConfigurationGeneratorChainConfigurationIDs	Returns generatorChainConfigurationIDs for the given designConfiguration element
getDesignConfigurationInterconnectionConfigurationIDs	Returns interconnectionConfigurationIDs for the given designConfiguration element
getInterconnectionConfigurationInterconnectionID	Returns interconnectionID for the given interconnectionConfiguration element
getInterconnectionConfigurationAbstractorInstantiationsIDs	Returns abstractorInstancesIDs for the given interconnectionConfiguration element
getAbstractorInstancesInterfaceComponentInstanceReference	Returns componentRef for the given abstractorInstancesInterface element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Name	Description
getAbstractorInstancesInterfaceBusInterfaceReference	Returns busRef for the given abstractorInstancesInterface element
getAbstractorInstancesAbstractorInstanceIDs	Returns the abstractorInstanceIDs for the given abstractorInstances element
getAbstractorInstanceName	Returns instanceName for the given abstractorInstance element
getAbstractorInstanceViewName	Returns viewName for the given abstractorInstance element
getDesignConfigurationViewConfigurationIDs	Returns viewConfigurationIDs for the given designConfiguration element
getDesignConfigurationViewConfigurationIDs	Returns instanceName for the given viewConfiguration element
getViewConfigurationViewID	Returns configured viewID for the given viewConfiguration element
createDesignConfiguration	Create a new designConfiguration with the given VLNV and returns its designID; fails and returns null if VLNV already exists
setDesignConfigurationDesignRef	Set designRef with the given VLNV for the given designConfiguration element
addDesignConfigurationGeneratorChainConfiguration	Add generatorChainConfiguration with the given VLNV to the given designConfiguration element
removeDesignConfigurationGeneratorChainConfiguration	Remove the given generatorChainConfiguration element
addDesignConfigurationInterconnectionConfiguration	Add interconnectionConfiguration with the given interconnectionRef, VLNV, instance, and viewName to the given designConfiguration element
removeDesignConfigurationInterconnectionConfiguration	Remove the given interconnectionConfiguration element
addInterconnectionConfigurationAbstractorInstances	Add abstractorInstances with the given VLNV, instance, and viewName to the given interconnectionConfiguration element
removeInterconnectionConfigurationAbstractorInstances	Remove the given abstractorInstances element
addAbstractorInstancesAbstractorInstance	Add abstractorInstance with the given VLNV, instance, and viewName to the given abstractorInstances element
removeAbstractorInstancesAbstractorInstance	Remove the given abstractorInstance element
addDesignConfigurationViewConfiguration	Add viewConfiguration with the given instanceName and viewName to the given designConfiguration element
removeDesignConfigurationViewConfiguration	Remove the given viewConfiguration element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.2.41 Design configuration operations (EXTENDED)

Name	Description
getAbstractorInstancesInterfaceIDs	Returns the abstractorInstancesInterfaceIDs for the given abstractorInstances element
getAbstractorInstanceID	Returns the abstractorInstanceID for the abstractorInstance element
addAbstractorInstancesInterface	Add abstractorInstancesInterface with the given componentRef and busRef to the given abstractorInstances element
removeAbstractorInstancesInterface	Remove the given abstractorInstancesInterface element

F.7.2.42 Design instantiation operations (BASE)

Name	Description
getDesignInstantiationDesignInstanceID	Returns a designInstanceID for the given designInstantiation element

F.7.2.43 Design operations (BASE)

Name	Description
getDesignComponentInstanceIDs	Returns componentInstanceIDs for the given design element
getComponentInstanceName	Returns name for the given componentInstance element
getDesignInterconnectionIDs	Returns interconnectionsIDs for the given design element
getInterconnectionActiveInterfaceIDs	Returns activeInterfaceIDs for the given interconnection element
getActiveInterfaceComponentInstanceReference	Returns componentRef for the given activeInterface element
getActiveInterfaceBusInterfaceReference	Returns busRef for the given activeInterface element
getActiveInterfaceExcludePorts	Returns excludePorts for the given activeInterface element
getInterconnectionHierInterfaceIDs	Returns hierInterfaceIDs for the given activeInterface element
getHierInterfaceBusInterfaceReference	Returns busRef for the given hierInterface element
getDesignMonitorInterconnectionIDs	Returns monitorInterconnectionIDs for the given design element
getMonitorInterconnectionMonitoredActiveInterfaceID	Returns monitorActiveInterfaceID for the given monitorInterconnection element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Name	Description
getMonitoredActiveInterfaceComponentInstanceReference	Returns componentRef for the given monitoredActiveInterface element
getMonitoredActiveInterfaceBusInterfaceReference	Returns busRef for the given monitoredActiveInterface element
getMonitoredActiveInterfacePath	Returns path for the given monitoredActiveInterface element
getMonitorInterconnectionMonitorInterfaceIDs	Returns monitorInterfaceIDs for the given monitorInterconnection element
getDesignAdHocConnectionIDs	Returns adHocConnectionsIDs for the given design element
getAdHocConnectionTiedValue	Returns tiedValue for the given adHocConnection element
getAdHocConnectionInternalPortReferenceIDs	Returns internalPortReferenceIDs for the given adHocConnection element
getAdHocInternalPortReferenceComponentInstanceReference	Returns componentRef for the given internalPortReference element
getAdHocInternalPortReferencePortReference	Returns portRef for the given internalPortReference element
getAdHocInternalPortReferenceRange	Returns range for the given internalPortReference element
getAdHocInternalPortReferenceIndices	Returns indices for the given internalPortReference element
getAdHocConnectionExternalPortReferenceIDs	Returns externalPortReferenceIDs for the given adHocConnection element
getAdHocExternalPortReferencePortReference	Returns componentRef for the given externalPortReference element
getAdHocExternalPortReferenceRange	Returns portRef for the given externalPortReference element
getAdHocExternalPortReferenceIndices	Returns range for the given externalPortReference element
createDesign	Create a new design with the given VLNV and returns its designID; fails and returns null if VLNV already exists
addDesignComponentInstance	Add componentInstance with the given VLNV and instance name to the given design element
removeDesignComponentInstance	Remove the given componentInstance element
addDesignInterconnection	Add interconnection with the given name, componentInstanceRef, and busRef to the given design element
removeDesignInterconnection	Remove the given interconnection element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Name	Description
addInterconnectionActiveInterface	Add activeInterface with the given componentInstanceRef and busRef tot the given interconnection element
removeInterconnectionActiveInterface	Remove the given activeInterface element
setActiveInterfaceExcludePorts	Set excludePorts with the given value for the given activeInterface element
addInterconnectionHierInterface	Add hierInterface with the given busRef to the given interconnection element
removeInterconnectionHierInterface	Remove the given hierInterface element
addDesignMonitorConnection	Add monitorInterconnection with the given name, componentInstanceRef, and busRef to the given design element
removeDesignMonitorConnection	Remove the given monitorInterconnection element
addMonitorInterconnectionMonitorInterface	Add monitorInterface with the given componentInstanceRef and busRef to the given monitorInterconnection element
removeMonitorInterconnectionMonitorInterface	Remove the given monitorInterface interface
addDesignAdHocConnection	Add adHocConnection with the given name, componentInstanceRef, and portRef to the given design element
addDesignExternalAdHocConnection	Add adHocConnection with the given name and portRef to the given design element
removeDesignAdHocConnection	Remove the given adHocConnection element
setAdHocConnectionTiedValue	Set tiedValue with the given value for the given adHocConnection element
addAdHocConnectionInternalPortReference	Add internalPortReference with the given componentInstanceRef and portRef to the given adHocConnection element
removeAdHocConnectionInternalPortReference	Remove the given internalPortReference element
setAdHocInternalPortReferenceRange	Set range with the given range for the given internalPortReference
setAdHocInternalPortReferenceIndices	Set indices with the given indices for the given internalPortReference
addAdHocConnectionExternalPortReference	Add externalPortReference with the portRef to the given adHocConnection element
removeAdHocConnectionExternalPortReference	Remove the given externalPortReference element
setAdHocExternalPortReferenceRange	Set range with the given range for the given externalPortReference
setAdHocExternalPortReferenceIndices	Set indices with the given indices for the given externalPortReference

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.2.44 Design operations (EXTENDED)

Name	Description
SetMonitoredActiveInterfacePath	Set path with the given value for the given monitoredActiveInterface element
SetMonitorInterfacePath	Set path with the given value for the given monitorInterface element

F.7.2.45 Driver operations (BASE)

Name	Description
getDriverRange	Returns the range of the given driver element
getDriverDefaultValue	Returns the default value of the given driver element
getDriverClockDriverID	Returns the clockDriverID of the given driver element
getClockDriverClockPeriodUnits	Returns the clockPeriodUnits of the given clockDriver element
getClockDriverClockPeriod	Returns the clockPeriod of the given clockDriver element
getClockDriverClockPulseOffsetUnits	Returns the clockPulseOffsetUnits of the given clockDriver element
getClockDriverClockPulseOffset	Returns the clockPulseOffset of the given clockDriver element
getClockDriverClockPulseValue	Returns the clockPulseValue of the given clockDriver element
getClockDriverClockPulseDurationUnits	Returns the clockPulseDurationUnits of the given clockDriver element
getClockDriverClockPulseDuration	Returns the clockPulseDuration of the given clockDriver element
getDriverSingleShotDriverID	Returns the singleShotDriverID of the given driver element
getSingleShotDriverSingleShotOffset	Returns the singleShotOffset of the given singleShotDriver element
getSingleShotDriverSingleShotValue	Returns the singleShotValue of the given singleShotDriver element
getSingleShotDriverSingleShotDuration	Returns the singleShotDuration of the given singleShotDriver element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.2.46 Driver operations (EXTENDED)

Name	Description
setDriverRange	Set the range of the given driver
addDriverDefaultValue	Add the given default value to the given driver element
removeDriverDefaultValue	Remove the default value from the given driver element
addDriverClockDriver	Add a clock driver with the given details to the given driver element
removeDriverClockDriver	Remove the clock driver from the given driver element
setClockDriverClockPeriodUnits	Set the units attribute of the clockPeriod element in the given clockDriver element
setClockDriverClockPulseOffsetUnits	Set the units attribute of the clockPulseOffset element in the given clockDriver element
setClockDriverClockPulseDurationUnits	Set the units attribute of the clockPulseDuration element in the given clockDriver element
addDriverSingleShotDriver	Add a single shot driver with the given details to the given driver element
removeDriverSingleShotDriver	Remove the single shot driver from the given driver element

F.7.2.47 Element attribute operations (BASE)

Name	Description
getAttributeValue	Returns value of the attribute with the given name in the given element
getAttributeValueExpression	Returns expression of the attribute with the given name in the given element

F.7.2.48 Element attribute operations (EXTENDED)

Name	Description
addAttribute	Add attribute with the given name and expression to the given element
removeAttribute	Remove attribute with the given name from the given element
setAttributeValue	Set attribute with the given name to the given expression in the given element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.2.49 Field operations (BASE)

Name	Description
getRegisterFieldID	Returns the value of the FieldID attribute in the given regField element

F.7.2.50 File builder operations (BASE)

Name	Description
getFileBuilderFileType	Return fileType for the given fileBuilder element
getFileBuilderCommand	Return command for the given fileBuilder element
getFileBuilderFlags	Return flags for the given fileBuilder element
getRegisterFieldID	Return replaceDefaultFlags for the given fileBuilder element

F.7.2.51 File builder operations (EXTENDED)

Name	Description
setFileBuilderCommand	Set command with the given value for the given fileBuilder element
setFileBuilderFlags	Set flags with the given value for the given fileBuilder element
setFileBuilderReplaceDefaultFlags	Set replaceDefaultFlags with the given value for the given fileBuilder element

F.7.2.52 File set operations (BASE)

Name	Description
getFileSetGroups	Returns groups for the given fileSet element
getFileSetFileIDs	Returns fileIDs for the given fileSet element
getFileFileTypes	Returns fileTypes for the given file element
getFileIsStructural	Returns isStructural for the given file element
getFileIsIncludeFile	Returns isIncludeFile for the given file element
getFileHasExternalDeclarations	Returns hasExternalDeclarations for the given file element
getFileLogicalName	Returns logicalName for the given file element
getFileExportedNames	Returns exportedNames for the given file element
getFileBuildCommand	Returns buildCommand for the given file element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Name	Description
getFileBuildCommandFlags	Returns buildCommandFlags for the given file element
getFileBuildCommandReplaceDefaultFlags	Returns buildCommandReplaceDefaultFlags for the given file element
getFileBuildCommandTargetName	Returns buildCommandTargetName for the given file element
getFileDependencies	Returns dependencies for the given file element
getFileDefineSymbolIDs	Returns fileDefineIDs for the given file element
getFileDefineSymbolValue	Returns value for the given fileDefine element
getFileImageTypes	Returns imageTypes for the given file element
getFileSetDefaultFileBuilderIDs	Returns fileBuilderIDs for the given fileSet element
getFileSetDependencies	Returns dependencies for the given fileSet element
getFileSetFunctionIDs	Returns functionsIDs for the given fileSet element
getFunctionReplicate	Returns replicate for the given function element
getFunctionEntryPoint	Returns entryPoint for the given function element
getFunctionFileID	Returns fileID for the given function element
getFunctionReturnType	Returns returnType the given function element
getFunctionArgumentIDs	Returns argumentIDs for the given function element
getFunctionArgumentDataType	Returns dataType for the given argumentID element
getFunctionDisabled	Returns disabled for the given function element
getFunctionSourceFileIDs	Returns sourceFileIDs for the given function element
getFunctionSourceFileName	Returns sourceName for the given functionSourceFile element
getFunctionSourceFileType	Returns fileType for the given functionSourceFile element
getFunctionSourceFileTypeUser	Returns user attribute for fileType for the given functionSourceFile element

F.7.2.53 File set operations (EXTENDED)

Name	Description
setFileSetGroups	Set groups with the given value for the given fileSet element
addFileSetFile	Add file with the given name and fileTypes to the given fileSet element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Name	Description
removeFileSetFile	Remove the given file
setFileIsStructural	Set isStructural with the given value for the given file element
setFileIsIncludeFile	Set isIncludeFile with the given value for the given file element
setFileHasExternalDeclarations	Set hasExternalDeclarations with the given value for the given file element
setFileLogicalName	Set logicalName with the given value for the given file element
setFileExportedNames	Set exportedNames with the given value for the given file element
setFileBuildCommand	Set buildCommand with the given value for the given file element
setFileBuildCommandFlags	Set buildCommandFlags with the given value for the given file element
setFileBuildCommandReplaceDefaultFlags	Set buildCommandReplaceDefaultFlags with the given value for the given file element
setFileBuildCommandTargetName	Set buildCommandReplaceDefaultFlags with the given value for the given file element
addFileDependency	Add given dependency to the given file element
removeFileSetDependency	Remove the given dependency from the given file element
addFileDefineSymbol	Add fileDefine with the given name and value to the given file element
removeFileDefineSymbol	Remove the given fileDefine element
setFileImageTypes	Set imageTypes with the given value for the given file element
addFileSetDefaultFileBuilder	Add defaultFileBuilder with the given fileType to the given file element
removeFileSetDefaultFileBuilder	Remove the given defaultFileBuilder element
addFileSetDependency	Add given dependency to the given fileSet element
removeFileSetDependency	Remove the given dependency from the given fileSet element
addFileSetFunction	Add function with the given fileRef to the given fileSet element
removeFileSetFunction	Remove the given function element
setFunctionReplicate	Set replicate to the given value for the given function element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Name	Description
setFunctionEntryPoint	Set entryPoint to the given value for the given function element
setFunctionReturnType	Set returnType to the given value for the given function element
addFunctionArgument	Add argument with given name and value to the given function element
removeFunctionArgument	Remove the given argument
setFunctionArgumentDataType	Set datatype with the given type for the given argument element
setFunctionDisabled	Set disabled to the given value for the given function element
addFunctionSourceFile	Add sourceFile with the given sourceName and fileType to the given function element
removeFunctionSourceFile	Remove the given sourceFile element
setFunctionSourceFileTypeUser	Set user attribute for the given sourceFile element

F.7.2.54 Generator chain operations (BASE)

Name	Description
getGeneratorChainGeneratorChainSelectorGroupSelectorIDs	Returns generatorChainSelectorGroupSelectorIDs for the given generatorChain element
getGroupSelectorSelectionOperator	Returns multipleGroupSelectionOperation for a given groupSelector element
getGroupSelectorSelectionNames	Returns names for a given groupSelector element
getGeneratorChainGeneratorChainSelectorGeneratorChainIDs	Returns generatorChainIDs for the given generatorChain element
getGeneratorChainComponentGeneratorSelectorGroupSelectorIDs	Returns generatorChainComponentGeneratorSelectorGroupSelectorIDs for the given generatorChain element
getGeneratorChainGeneratorIDs	Returns generatorIDs for the given generatorChain element
getGeneratorChainGroups	Returns chainGroups for the given generatorChain element
getGeneratorChainChoiceIDs	Returns choiceIDs for the given generatorChain element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.2.55 Generator chain operations (EXTENDED)

Name	Description
createGeneratorChain	Create a new generatorChain with the given VLNV and returns its generatorChainID; fails and returns null if VLNV already exists
addGeneratorChainGeneratorChainSelectorGroupSelector	Add generatorChainMultipleGroupSelector with the given names for a given generatorChain element
setGroupSelectorSelectionOperator	Set multipleGroupSelectionOperation with the given value for a given groupSelector element
addGeneratorChainComponentGeneratorSelectorGroupSelector	Add generatorChainComponentGeneratorSelectorGroupSelector with the given names for the given generatorChain element
addGeneratorChainGenerator	Add generator with the given generatorExe to the given generatorChain element
addGeneratorChainGroup	Add chainGroup with the given value to the given generatorChain element
removeGeneratorChainGroup	Remove chainGroup with the given value from the given generatorChain element
addGeneratorChainChoice	Add choice with the given name and enumerations to the given generatorChain element
removeGeneratorChainChoice	Remove the given choice element

F.7.2.56 Generator operations (BASE)

Name	Description
getGeneratorHidden	Returns hidden for the given generator element
getGeneratorScope	Returns scope for the given generator element
getGeneratorPhase	Returns phase for the given generator element
getGeneratorApiType	Returns apiType for the given generator element
getGeneratorTransportMethods	Returns transportMethods for the given generator element
getGeneratorExecutable	Returns generatorExe for the given generator element
getGeneratorGroups	Returns group names for the given generator element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.2.57 Generator operations (EXTENDED)

Name	Description
setGeneratorHidden	Set hidden with the given value for the given generator element
setGeneratorScope	Set scope with the given value for the given generator element
setGeneratorPhase	Set phase with the given value for the given generator element
setGeneratorApiType	Set apiType with the given value for the given generator element
setGeneratorTransportMethods	Set transportMethods with the given value for the given generator element
setGeneratorExecutable	Set generatorExe with the given value for the given generator element
getAbstractionDefIDs	Set groups with the given value for the given generator element

F.7.2.58 Global operations (BASE)

Name	Description
getAbstractionDefIDs	Returns all registered abstractionDefIDs
getBusDefIDs	Returns all registered busDefIDs
getComponentIDs	Returns all registered componentIDs
getDesignIDs	Returns all registered designIDs
getDesignConfigurationIDs	Returns all registered designConfigurationIDs
getAbstractorIDs	Returns all registered abstractorIDs
getGeneratorChainIDs	Returns all registered generatorChainIDs
getCatalogIDs	Returns all registered catalogIDs
getDesignID	Returns the designID of the current or top design associated with the currently invoked generator
getGeneratorContextComponentInstanceID	Returns the componentInstanceID associated with the currently invoked generator
getXML	Returns XML fragment
message	Send message level and message text to DE
init	API initialization function Must be called before any other API call

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Name	Description
end	Terminate connection to the DE
save	Save all edits done in generator to DE; document must be valid
registerVLNV	Register the VLNV contained in the given file, possibly replacing an existing VLNV
unregisterVLNV	Unregister the given VLNV
registerCatalogVLNVs	Register all VLNVs in the given catalog
unregisterCatalogVLNVs	Unregister all VLNVs in the given catalog

F.7.2.59 Indirect interface operations (BASE)

Name	Description
getIndirectInterfaceIndirectAddress	Returns indirectAddress for the given indirectInterface element
getIndirectInterfaceIndirectData	Returns indirectData for the given indirectInterface element
getIndirectInterfaceMemoryMapID	Returns memoryMapID for the given indirectInterface element
getIndirectInterfaceBridgeIDs	Returns bridgeIDs for the given indirectInterface element
getIndirectInterfaceBitSteering	Returns indirectAddress for the given indirectInterface element
getIndirectInterfaceEndianness	Returns indirectAddress for the given indirectInterface element

F.7.2.60 Indirect interface operations (EXTENDED)

Name	Description
addIndirectInterfaceBridge	Add bridge with the given busInterfaceRef to the given indirectInterface element
removeIndirectInterfaceBridge	Remove the given bridge element
setIndirectInterfaceBitSteering	Set the bitSteering with the given value for the given indirectInterface element
setIndirectInterfaceEndianness	Set the endianness with the given value for the given indirectInterface element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.2.61 Memory map operations (BASE)

Name	Description
getMemoryMapElementIDs	Returns the memoryMapElementIDs for the given memoryMap or localMemoryMap element
getMemoryMapElementType	Returns the elementType for the given memoryMapElement element
getAddressBlockBaseAddress	Returns the baseAddress for the given addressBlock element
getAddressBlockTypeIdentifier	Returns the typeIdentifier for the given addressBlock element
getAddressBlockRange	Returns the range for the given address Block element
getAddressBlockWidth	Returns the width for the given address Block element
getAddressBlockUsage	Returns the usage for the given address Block element
getAddressBlockVolatility	Returns the volatility for the given address Block element
getAddressBlockAccess	Returns the access for the given address Block element
getAddressBlockRegisterIDs	Returns the registerIDs for the given address Block element
getAddressBlockRegisterFileIDs	Returns the registerFileIDs for the given address Block element
getBankAlignment	Returns the value of the bankAlignment attribute for the given bank or localBank element
getBankBaseAddress	Returns the baseAddress for the given bank or localBank element
getBankBlockAndSubspaceElementIDs	Returns the bankIDs, addressBlockIDs, and subSpaceMapIDs for the given bank or localBank element
getBankUsage	Returns the usage for the given bank or localBank element
getBankVolatility	Returns the volatility for the given bank or localBank element
getBankAccess	Returns the access for the given bank or localBank element
getSubspaceMapMasterID	Returns an associated busInterfaceID of the masterRef for the given subspaceMap element
getSubspaceMapSegmentID	Returns an associated segmentID of the segmentRef for the given subspaceMap
getSubspaceMapBaseAddress	Returns the baseAddress for the given subspaceMap
getMemoryMapRemapIDs	Returns the memoryRemapIDs for the given memoryMap or localMemoryMap element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Name	Description
getMemoryRemapState	Returns the state for a given memoryRemap element
getMemoryRemapElementIDs	Returns the bankIDs, addressBlockIDs, and subSpaceMapIDs for the given memoryRemap element
getMemoryMapAddressUnitBits	Returns the addressUnitBits for the given memoryMap or localMemoryMap element
getMemoryMapShared	Returns the shared for the given memoryMap or localMemoryMap element

F.7.2.62 Memory map operations (EXTENDED)

Name	Description
addMemoryMapAddressBlock	Add an addressBlock with the given name, baseAddress, range, and width to the given memoryMap or localMemoryMap element
removeMemoryMapAddressBlock	Remove the given addressBlock element
setAddressBlockRange	Set the range with the given value for the given addressBlock element
setAddressBlockTypeIdentifier	Set the typeIdentifier with the given value for the given addressBlock element
setAddressBlockUsage	Set the usage with the given value for the given addressBlock element
setAddressBlockVolatility	Set the volatility with the given value for the given addressBlock element
setAddressBlockAccess	Set the access with the given value for the given addressBlock element
addAddressBlockRegister	Add a register with the given name, offset, and size, and a field with the given name, offset, and width to the given addressBlock
removeAddressBlockRegister	Remove the given register element
addAddressBlockRegisterFile	Add a registerFile with the given name, addressOffset, and range to the given addressBlock element
removeAddressBlockRegisterFile	Remove the given registerFile element
addMemoryMapBank	Add a bank with the given name, baseAddress, and alignment to the given memoryMap or localMemoryMap element
removeMemoryMapBank	Remove the given bank element
addBankBank	Add a bank with the given name, baseAddress, and alignment to the given bank element
removeBankBank	Remove the given bank element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Name	Description
setBankUsage	Set usage with the given value for the given bank element
setBankVolatility	Set volatility with the given value for the given bank element
setBankAccess	Set access with the given value for the given bank element
addMemoryMapSubspaceMap	Add a subspaceMap with the given name, masterRef, and baseAddress to the given memory map
removeMemoryMapSubspaceMap	Remove the given subspaceMap element
setMemoryRemapSegmentRef	Set the segmentRef with the given value for the given subspaceMap element
addMemoryMapRemapWithAddressBlock	Add a memoryRemap with the given state and name containing an addressBlock with the given addressBlock name, baseAddress, range, and width to the given memoryMap or localMemoryMap element
addMemoryMapRemapWithBank	Add a memoryRemap with the given state and name containing a bank with the given bankName, baseAddress, and alignment to the given memoryMap or localMemoryMap element
addMemoryMapRemapWithSubspaceMap	Add a memoryRemap with the given state and name containing a subspaceMap with the given subspaceName, baseAddress, and masterRef to the given memoryMap element
removeMemoryMapRemap	Remove the given memoryRemap element
setMemoryMapShared	Set the shared with the given value for the given memoryMap element

F.7.2.63 Module and type parameter operations (BASE)

Name	Description
getModuleOrTypeParameterIDs	Returns moduleOrTypeParameterIDs of the given element
getModuleOrTypeParameterValue	Returns value of the given moduleOrTypeParameter element
getModuleOrTypeParameterValueExpression	Returns expression of the given moduleOrTypeParameter element
getModuleOrTypeParameterDataType	Returns the data type value of the given moduleOrTypeParameter element
getModuleOrTypeParameterUsageType	Returns the usage value of the given moduleOrTypeParameter element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.2.64 Module and type parameter operations (EXTENDED)

Name	Description
addModuleOrTypeParameter	Add a moduleOrTypeParameter with the given name and given value to the given element
removeModuleOrTypeParameter	Remove the given moduleOrTypeParameter
setModuleOrTypeParameterValue	Set the value of the given moduleOrTypeParameter
setModuleOrTypeParameterDataType	Set the data type value of the given moduleOrTypeParameter
setModuleOrTypeParameterUsageType	Set the usage value of the given moduleOrTypeParameter

F.7.2.65 Parameter operations (BASE)

Name	Description
getParameterIDs	Returns parameterIDs of the given element
getParameterValue	Returns value of the given parameter element
getParameterValueExpression	Returns expression of the given parameter element
addParameter	Add a parameter with the given name and given value to the given element
removeParameter	Remove the given parameter
setParameterValue	Set the value of the given parameter

F.7.2.66 Port operations (BASE)

Name	Description
getPortStyle	Returns the style (wire or transactional) for the given port element
getPortAllLogicalDirectionsAllowed	Returns allLogicalDirectionsAllowed for the given port element
getPortDirection	Returns direction for the given port element
getPortWireTypeDefIDs	Returns wireTypeDefIDs for the given port element
getPortDriverIDs	Returns driverIDs for the given port element
getPortConstraintSetIDs	Returns constraintSetIDs or the given port element
getPortInitiative	Returns initiative for the given port element
getPortKind	Returns kind for the given port element
getPortKindCustom	Returns custom for the given port element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Name	Description
getPortBusWidth	Returns busWidth for the given port element
getPortProtocolType	Returns protocolType for the given port element
getPortProtocolTypeCustom	Returns custom attribute of protocolType the given port element
getPortProtocolPayloadName	Returns payloadName for the given port element
getPortProtocolPayloadType	Returns payloadType for the given port element
getPortProtocolPayloadExtension	Returns payloadExtension for the given port element
getPortProtocolPayloadExtensionMandatory	Returns mandatory attribute of payloadExtension for the given port element
getPortTransTypeDefIDs	Returns transTypeDefIDs for the given port element
getPortMaxConnections	Returns maxConnections for the given port element
getPortMinConnections	Returns minConnections for the given port element
getPortAccessType	Returns accessType for the given port element

F.7.2.67 Port operations (EXTENDED)

Name	Description
setPortAllLogicalDirectionsAllowed	Set allLogicalDirectionsAllowed with the given value for the given port element
addPortWireTypeDef	Add wireTypeDef for the given port element
removePortWireTypeDef	Remove the given wireTypeDef element
addPortDefaultDriver	Add defaultDriver with the given value to the given port element
removePortDefaultDriver	Remove the given defaultDriver element
addPortClockDriver	Add clockDriver with the given clock period, pulse offset, pulse value, and pulse duration to the given port element
removePortClockDriver	Remove the given clockDriver element
addPortSingleShotDriver	Add singleShotDriver with the given offset, value, and duration to the given port element
removePortSingleShotDriver	Remove the given singleShort driver element
addPortConstraintSet	Add constraintSet to the given port element
removePortConstraintSet	Remove the given constraintSet element
setPortKind	Set kind with the given value for the given port element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Name	Description
setPortKindCustom	Set attribute custom of kind with the given value for the given port element
setPortBusWidth	Set busWidth with the given value for the given port element
setPortProtocolType	Set protocolType with the given value for the given port element
setPortProtocolTypeCustom	Set attribute custom of protocolType with the given value for the given port element
addPortProtocolPayload	Add a payload with the given type to the given port element
removePortProtocolPayload	Remove the given payload
setProtocolPayloadName	Set payload name with the given value for the given port element
setProtocolPayloadExtension	Set payload extension with the given value for the given port element
setProtocolPayloadExtensionMandatory	Set attribute mandatory of payload extension with the given value for the given port element
addPortTransTypeDef	Add transTypeDef for the given port element
removePortTransTypeDef	Remove the given transTypeDef element
setPortMaxConnections	Set maxConnections with the given value for the given port element
setPortMinConnections	Set minConnections with the given value for the given port element
setPortAccessType	Set portAccessType with the given value for the given port element

F.7.2.68 Present operations (BASE)

Name	Description
getIsPresentValue	Returns the value of the isPresent element of the given element
getIsPresentValueExpression	Returns the expression of the isPresent element of the given element
setIsPresentValue	Set the expression of the isPresent element of the given element
addIsPresent	Add an isPresent element to the given element with the given expression
removeIsPresent	Remove the isPresent element from the given element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.2.69 Register file operations (BASE)

Name	Description
getRegisterFileID	Returns the value of the RegisterFileID attribute in the given registerFile element
getRegisterFileDimensions	Returns the dim in the given registerFile element
getRegisterFileAddressOffset	Returns the addressOffset in the given registerFile element
getRegisterFileTypeIdentifier	Returns the typeIdentifier in the given registerFile element
setRegisterFileID	Returns the range in the given registerFile element

F.7.2.70 Register file operations (EXTENDED)

Name	Description
setRegisterFileID	Set the value of the RegisterFileID attribute in the given registerFile element
setRegisterFileDimensions	Set the dim with the given value n the given registerFile element
getRegisterFileTypeIdentifier	Set the typeIdentifier with the given value n the given registerFile element

F.7.2.71 Register operations (BASE)

Name	Description
getRegisterDimensions	Returns dim for the given register element
getRegisterAddressOffset	Returns addressOffset for the given register element
getRegisterTypeIdentifier	Returns typeIdentifier for the given register element
getRegisterSize	Returns size for the given register element
getRegisterVolatility	Returns volatility for the given register element
getRegisterAccess	Returns access for the given register element
getRegisterFieldIDs	Returns fieldIDs for the given register element
getRegisterAlternateRegisterIDs	Returns alternativeRegisterIDs for the given register element
getAlternateRegisterState	Returns state for the given alternateRegister element
getAlternateRegisterGroups	Returns groups for the given alternateRegister element
getAlternateRegisterTypeIdentifier	Returns typeIdentifier for the given alternateRegister element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Name	Description
getAlternateRegisterVolatility	Returns volatility for the given alternateRegister element
getAlternateRegisterAccess	Returns access for the given alternateRegister element
getAlternateRegisterFieldIDs	Returns fieldIDs for the given alternateRegister element
getRegisterResetValue	Returns resetValue derived from fieldResetValues for the given resetType and register element
getRegisterResetMask	Returns resetMask derived from fieldResetMasks for the given resetType and register element
getAlternateRegisterResetValue	Returns resetValue derived from fieldResetValues for the given resetType and alternateRegister element
getAlternateRegisterResetMask	Returns resetMask derived from fieldResetMasks for the given resetType and alternateRegister element

F.7.2.72 Register operations (EXTENDED)

Name	Description
setRegisterDimensions	Set dim with the given value for the given register element
setRegisterTypeIdentifier	Set typeIdentifier with the given value for the given register element
setRegisterVolatility	Set volatility with the given value for the given register element
setRegisterAccess	Set access with the given value for the given register element
addRegisterField	Add regField with the given name, offset, and width to the given register element
removeRegisterField	Remove the given regField element
addRegisterAlternateRegister	Add alternateRegister with the given groups to the given register element
removeRegisterAlternateRegister	Remove the given alternateRegister element
setAlternateRegisterState	Set state with the given value for the given alternateRegister element
setAlternateRegisterTypeIdentifier	Set typeIdentifier with the given value for the given alternateRegister element
setAlternateRegisterVolatility	Set volatility with the given value for the given alternateRegister element
setAlternateRegisterAccess	Set access with the given value for the given alternateRegister element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Name	Description
addAlternateRegisterField	Add regField with the given name, offset, and width to the given alternateRegister element
removeAlternateRegisterField	Remove the given regField element
setRegisterResetValue	Set the fieldResetValues with the given register reset value expression for the given resetType and register element
setRegisterResetMask	Set the fieldResetMasks with the given register reset mask expression for the given resetType and register element
setAlternateRegisterResetValue	Set the fieldResetValues with the given alternateRegister reset value expression for the given resetType and alternateRegister element
setAlternateRegisterResetMask	Set the fieldResetMasks with the given alternateRegister reset mask expression for the given resetType and alternateRegister element

F.7.2.73 Remap state operations (EXTENDED)

Name	Description
getRemapStateRemapPortIDs	Returns remapPortIDs for the given remapState element
getRemapPortPortID	Returns the portID of the portRef for the given remapPort element
getRemapPortPortIndex	Returns the portIndex for the given remapPort element
getRemapPortValue	Returns the value for the given remapPort element
addRemapStateRemapPort	Add a remapPort with the given portRef and value to the given remapState element
removeRemapStateRemapPort	Remove the given remapPort element
setRemapPortPortIndex	Set the portIndex with the given value for the given remapPort

F.7.2.74 Slice operations (BASE)

Name	Description
getSlicePathSegmentIDs	Returns the pathSegmentIDs for a given slice element
getPathSegmentName	Returns the pathSegmentName for a given pathSegment element
getPathSegmentIndices	Returns the indices for a given pathSegment element
getSliceRange	Returns the range for a given slice element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.2.75 Slice operations (EXTENDED)

Name	Description
addSlicePathSegment	Add a pathSegment with the given pathSegmentNames and indices to the given slice element
removeSlicePathSegment	Remove the given pathSegment element
setSliceRange	Set the range to the given range for the given slice element

F.7.2.76 Typedef operations (BASE)

Name	Description
getTypeDefTypeName	Returns the typeName for the given typeDef element
getTypeDefTypeDefinitions	Returns the typeDefinitions for the given typeDef element
getTypeDefTypeParameterIDs	Returns the typeParameterIDs for the given transTypeDef or serviceTypeDef element
getTypeDefServiceTypeDefIDs	Returns the serviceTypeDefIDs for the given transTypeDef element
getTypeDefViewIDs	Returns the viewIDs for the given typeDef element

F.7.2.77 Typedef operations (EXTENDED)

Name	Description
setTypeDefTypeName	Set the typeName with the given value for the given typeDef element
setTypeDefTypeDefinitions	Set the typeDefinitions with the given definitions for the given typeDef element
addTypeDefTypeParameter	Add a typeParameter with the given name and value to the given typeDef element
removeTypeDefTypeParameter	Remove the given typeParameter
addTypeDefServiceTypeDef	Add a serviceTypeDef with the given typeName to the given typeDef element
removeTypeDefServiceTypeDef	Remove the given serviceTypeDef
addTypeDefViewName	Add the given viewName to the viewRefs in the given typeDef element
removeTypeDefViewName	Removed the given viewName from the viewRefs in the given typeDef element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.2.78 Vector operations (BASE)

Name	Description
getVectorIDs	Returns vectorIDs of the given element
getVectorRange	Returns range of the given vector element
getVectorRangeExpression	Returns the range expressions of the given vector element

F.7.2.79 Vector operations (EXTENDED)

Name	Description
addVector	Add a vector to the given element
removeVector	Remove the given vector

F.7.2.80 Vendor extensions operations (BASE)

Name	Description
getVendorExtensions	Returns vendorExtensionID of the given element
addVendorExtensions	Add a vendor extension with given value to the given element
removeVendorExtensions	Remove the given vendor extension
setVendorExtensions	Set the value of the given vendor extension

F.7.2.81 View operations (BASE)

Name	Description
getViewEnvIdentifier	Returns envIdentifier for the given view element
getViewComponentInstantiationRef	Returns componentInstantiationRef for the given view element
getViewComponentInstantiationID	Returns componentInstantiationID for the given view element
getViewDesignInstantiationRef	Returns designInstantiationRef for the given view element
getViewDesignInstantiationID	Returns designInstantiationID for the given view element
getViewDesignConfigurationInstantiationRef	Returns designConfigurationInstantiationRef for the given view element
getViewDesignConfigurationInstantiationID	Returns designConfigurationInstantiationID for the given view element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.2.82 View operations (EXTENDED)

Name	Description
setViewComponentInstantiationRef	Set componentInstantiationRef for the given view element
setViewDesignInstantiationRef	Set designInstantiationRef for the given view element
setViewDesignConfigurationInstantiationRef	Set designConfigurationInstantiationRef for the given view element

F.7.2.83 Whitebox operations (BASE)

Name	Description
getWhiteboxElementType	Returns whiteboxType for the given whiteboxElement
getWhiteboxElementDrivable	Returns driveable for the given whiteboxElement

F.7.2.84 Whitebox operations (EXTENDED)

Name	Description
setWhiteboxElementDrivable	Set driveable with the given value for the given whiteBox element

F.7.2.85 Whitebox ref operations (BASE)

Name	Description
getWhiteboxElementRefID	Returns the value of the attribute name in the given whiteboxElementRef element
getWhiteboxElementRefLocationIDs	Returns the whiteboxElementReflocationIDs for the given whiteboxElementRef element
getWhiteboxElementRefLocationSlicesIDs	Returns the slideIDs for the given whiteboxElementRefLocationID

F.7.2.86 Whitebox ref operations (EXTENDED)

Name	Description
addWhiteboxElementRefLocation	Adds a whiteboxElementRefLocation element to a given whiteboxElementRef element
removeWhiteboxElementRefLocation	Removes the given whiteboxElementRefLocation element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Name	Description
addWhiteboxElementRefLocationSlice	Adds a slice to a given whiteboxElementRefLocation element
removeWhiteboxElementRefLocationSlice	Removes the given slice element

F.7.2.87 name group operations (BASE)

Name	Description
getName	Returns name of the given element
getDisplayName	Returns display name of the given element
getDescription	Returns description of the given element
addDisplayName	Add given display name to the given element; fails if display name is already set
removeDisplayName	Remove display name from the given element; fails if display name is not already set
setDisplayName	Set given display name for the given element; fails if display name is not already set
addDescription	Add given description to the given element; fails if description is already set
removeDescription	Remove description from the given element; fails if description is not already set
setDescription	Set given description for the given element; fails if description is not already set

F.7.2.88 name group operations (EXTENDED)

Name	Description
setName	Set name of the given element

F.7.2.89 top element operations (BASE)

Name	Description
getID	Returns ID of the element with the given VLNV or null if this element does not exist
getVLNV	Returns VLNV of the element with the given ID
getXMLPath	Returns location of the XML file that is registered with the VLNV of the given top-element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Name	Description
edit	Signal start of editing of the given top-element
setXMLPath	Set new location for the XML file that is registered with the VLNV of the given top-element

F.7.3 (BASE)

F.7.3.1 getRegisterFieldBitOffset

Description: Returns the bitOffset in the given regField element

- Returns: `offset` of type ***Number*** - Field bit offset
- Input: `regFieldID` of type ***String*** - Handle to a regField element

F.7.3.2 getRegisterFieldResetIDs

Description: Returns the resetIDs in the given regField element

- Returns: `resetIDs` of type ***String[]*** - List of reset handles
- Input: `regFieldID` of type ***String*** - Handle to a regField element

F.7.3.3 getResetTypeRef

Description: Returns the resetTypeRef for the given reset element

- Returns: `typeRef` of type ***String*** - Reset type
- Input: `resetID` of type ***String*** - Handle to a reset element

F.7.3.4 getResetValue

Description: Returns the value for the given reset element

- Returns: `value` of type ***Number*** - Reset value
- Input: `resetID` of type ***String*** - Handle to a reset element

F.7.3.5 getResetValueExpression

Description: Returns the value expression for the given reset element

- Returns: `expression` of type ***String*** - Reset value expression
- Input: `resetID` of type ***String*** - Handle to a reset element

F.7.3.6 getResetMask

Description: Returns the mask for the given reset element

- Returns: `mask` of type ***Number*** - Reset mask
- Input: `resetID` of type ***String*** - Handle to a reset element

F.7.3.7 getResetMaskExpression

Description: Returns the mask expression for the given reset element

- Returns: `expression` of type ***String*** - Reset mask expression

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

— Input: `resetID` of type ***String*** - Handle to a reset element

F.7.3.8 getRegisterFieldTypeIdentifier

Description: Returns the typeIdentifier for the given regField element

— Returns: `typeIdentifier` of type ***String*** - Field type identifier

— Input: `regFieldID` of type ***String*** - Handle to a regField element

F.7.3.9 getRegisterFieldBitWidth

Description: Returns the bitWidth for the given regField element

— Returns: `width` of type ***Number*** - Field bit width

— Input: `regFieldID` of type ***String*** - Handle to a regField element

F.7.3.10 getRegisterFieldVolatility

Description: Returns the volatility or the given regField element

— Returns: `volatile` of type ***Boolean*** - Field volatility

— Input: `regFieldID` of type ***String*** - Handle to a regField element

F.7.3.11 getRegisterFieldAccess

Description: Returns the access or the given regField element

— Returns: `access` of type ***String*** - Field access

— Input: `regFieldID` of type ***String*** - Handle to a regField element

F.7.3.12 getRegisterFieldEnumeratedValueIDs

Description: Returns the enumreatedValueIDs for the given regField element

— Returns: `enumeratedValueIDs` of type ***String[]*** - List of enumeratedValue handles

— Input: `regFieldID` of type ***String*** - Handle to a regField element

F.7.3.13 getEnumeratedValueUsage

Description: Returns the usage for the given enumeratedValue element

— Returns: `usage` of type ***String*** - EnumeratedValue usage

— Input: `enumeratedValueID` of type ***String*** - Handle to a enumeratedValue element

F.7.3.14 getEnumeratedValueValue

Description: Returns the value for the given enumeratedValue element

— Returns: `value` of type ***Number*** - EnumeratedValue value

— Input: enumeratedValueID of type ***String*** - Handle to a enumeratedValue element

F.7.3.15 getRegisterFieldModifiedWriteValue

Description: Returns the modifiedWriteValue for the given regField element

— Returns: `type` of type ***String*** - Field modifiedWriteValue type

— Input: `regFieldID` of type ***String*** - Handle to a regField element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.3.16 getRegisterFieldModifiedWriteValueModify

Description: Returns the modify attribute value of modifiedWriteValue for the given regField element

- Returns: `modify` of type ***String*** - Field modifiedWriteValue modify
- Input: `regFieldID` of type ***String*** - Handle to a regField element

F.7.3.17 getRegisterFieldWriteValueConstraintWriteAsRead

Description: Returns the writeAsRead for the given regField element

- Returns: `value` of type ***Boolean*** - Field writeAsRead value
- Input: `regFieldID` of type ***String*** - Handle to a regField element

F.7.3.18 getRegisterFieldWriteValueConstraintUseEnumeratedValues

Description: Returns the useEnumeratedValues for the given regField element

- Returns: `value` of type ***Boolean*** - Field useEnumatedValues value
- Input: `regFieldID` of type ***String*** - Handle to a regField element

F.7.3.19 getRegisterFieldWriteValueConstraintMinMax

Description: Returns the min and max for the given regField element

- Returns: `minmax` of type ***Number[]*** - Field min at index 0 and max at index 1 value
- Input: `regFieldID` of type ***String*** - Handle to a regField element

F.7.3.20 getRegisterFieldReadAction

Description: Returns the readAction for the given regField element

- Returns: `type` of type ***String*** - Field readAction type
- Input: `regFieldID` of type ***String*** - Handle to a regField element

F.7.3.21 getRegisterFieldReadActionModify

Description: Returns the modify attribute value of readAction for the given regField element

- Returns: `modify` of type ***String*** - Field readAction modify
- Input: `regFieldID` of type ***String*** - Handle to a regField element

F.7.3.22 getRegisterFieldTestable

Description: Returns the testAble for the given regField element

- Returns: `value` of type ***Boolean*** - Field testable value
- Input: `regFieldID` of type ***String*** - Handle to a regField element

F.7.3.23 getRegisterFieldTestContraint

Description: Returns the testConstraint for the given regField element

- Returns: `value` of type ***String*** - Field testContraint value
- Input: `regFieldID` of type ***String*** - Handle to a regField element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.3.24 getRegisterFieldReserved

Description: Returns the reserved for the given regField element

- Returns: `value` of type ***Boolean*** - Field reserved value
- Input: `regFieldID` of type ***String*** - Handle to a regField element

F.7.4 (EXTENDED)

F.7.4.1 addRegisterFieldReset

Description: Add a reset with the given type and value to the given regField element

- Returns: `resetID` of type ***String*** - Handle to a new reset element
- Input: `regFieldID` of type ***String*** - Handle to a regField element
- Input: `typeRef` of type ***String*** - Reset type name
- Input: `value` of type ***String*** - Reset value expression

F.7.4.2 removeRegisterFieldReset

Description: Remove the given reset element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `resetID` of type ***String*** - Handle to a reset element

F.7.4.3 setResetMask

Description: Set the mask with the given value for the given reset element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `resetID` of type ***String*** - Handle to a reset element
- Input: `mask` of type ***String*** - Reset mask expression

F.7.4.4 setRegisterFieldID

Description: Set the FieldID attribute to the given value for the given regField element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `regFieldID` of type ***String*** - Handle to a regField element
- Input: `id` of type ***String*** - FieldID attribute value

F.7.4.5 setRegisterFieldTypeIdentifier

Description: Set the typeIdentifier to the given value for the given regField element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `regFieldID` of type ***String*** - Handle to a regField element
- Input: `typeIdentifier` of type ***String*** - RegField typeIdentifier

F.7.4.6 setRegisterFieldVolatility

Description: Set the volatility to the given value for the given regField element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `regFieldID` of type ***String*** - Handle to a regField element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

— Input: `volatile` of type ***Boolean*** - RegField volatility

F.7.4.7 setRegisterFieldAccess

Description: Set the access to the given value for the given regField element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `regFieldID` of type ***String*** - Handle to a regField element
— Input: `access` of type ***String*** - RegField access

F.7.4.8 addRegisterFieldEnumeratedValue

Description: Add a enumeratedValue with the given name and value to the given regField element

— Returns: `enumeratedValueID` of type ***String*** - Handle to a new enumeratedValue element
— Input: `regFieldID` of type ***String*** - Handle to a regField element
— Input: `name` of type ***String*** - EnumeratedValue name
— Input: `value` of type ***String*** - EnumeratedValue value

F.7.4.9 removeRegisterFieldEnumeratedValue

Description: Remove the given enumeratedValue element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `enumeratedValueID` of type ***String*** - Handle to a enumeratedValue element

F.7.4.10 setRegisterFieldEnumeratedValueUsage

Description: Set the usage with the given value for the given enumeratedValue element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `enumeratedValueID` of type ***String*** - Handle to a enumeratedValue element
— Input: `usage` of type ***String*** - EnumeratedValue usage

F.7.4.11 setRegisterFieldModifiedWriteValue

Description: Set the modifiedWriteValue with the given type for the given regField element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `regFieldID` of type ***String*** - Handle to a regField element
— Input: `type` of type ***String*** - ModifiedWriteValue type

F.7.4.12 setRegisterFieldModifiedWriteValueModify

Description: Set the modify attribute of modifiedWriteValue with the given value for the given regField element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `regFieldID` of type ***String*** - Handle to a regField element
— Input: `modify` of type ***String*** - ModifiedWriteValue modify

F.7.4.13 setRegisterFieldWriteValueContraint

Description: Set the writeValueContraint elements writeAsRead, useEnumeratedValues, min, and max with the given values for the given regField element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `regFieldID` of type ***String*** - Handle to a regField element
- Input: `writeAsRead` of type ***Boolean*** - WriteValueConstraint writeAsRead or null
- Input: `useEnumeratedValues` of type ***Boolean*** - WriteValueConstraint useEnumeratedValues or null
- Input: `minmax` of type ***Number[]*** - WriteValueConstraint minmax or null

F.7.4.14 setRegisterFieldReadAction

Description: Set the readAction with the given type for the given regField element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `regFieldID` of type ***String*** - Handle to a regField element
- Input: `type` of type ***String*** - ReadAction type

F.7.4.15 setRegisterFieldReadActionModify

Description: Set the modify attribute of readAction with the given value for the given regField element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `regFieldID` of type ***String*** - Handle to a regField element
- Input: `modify` of type ***String*** - ReadAction modify

F.7.4.16 setRegisterFieldTestable

Description: Set the testable with the given value for the given regField element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `regFieldID` of type ***String*** - Handle to a regField element
- Input: `value` of type ***Boolean*** - Testable value

F.7.4.17 setRegisterFieldTestContraint

Description: Set the testConstraint with the given value for the given regField element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `regFieldID` of type ***String*** - Handle to a regField element
- Input: `value` of type ***String*** - TestContraint value

F.7.4.18 setRegisterFieldReserved

Description: Set the reserved with the given value for the given regField element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `regFieldID` of type ***String*** - Handle to a regField element
- Input: `value` of type ***Boolean*** - Reserved value

F.7.5 Abstraction definition operations (BASE)

F.7.5.1 getAbstractionDefinitionBusType

Description: Returns busType for the given abstractionDef or abstractionDefInstance element

- Returns: `busDefVLNV` of type ***String[]*** - BusDef VLNV

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

— Input: `abstractionDefID` | `abstractionDefInstanceID` of type ***String*** - Handle to an abstractionDef or abstractionDefInstance element

F.7.5.2 getAbstractionDefinitionExtends

Description: Returns extends for the given abstractionDef or abstractionDefInstance element

— Returns: `abstractionDefVLNV` of type ***String*** - AbstractionDef VLNV

— Input: `abstractionDefID` | `abstractionDefInstanceID` of type ***String*** - Handle to an abstractionDef or abstractionDefInstance element

F.7.5.3 getAbstractionDefinitionPortIDs

Description: Returns abstractionDefPortIDs for the given abstractionDef or abstractionDefInstance element

— Returns: `abstractionDefPortIDs` of type ***String[]*** - List of abstractionDefPort handles

— Input: `abstractionDefID` | `abstractionDefInstanceID` of type ***String*** - Handle to an abstractionDef or abstractionDefInstance element

F.7.5.4 getAbstractionDefPortLogicalName

Description: Returns logicalName for the given abstractionDefPort element

— Returns: `logicalName` of type ***String*** - Logical port name

— Input: `abstractionDefPortID` of type ***String*** - Handle to abstractionDefPort element

F.7.5.5 getAbstractionDefPortStyle

Description: Returns portStyle for the given abstractionDefPort element

— Returns: `style` of type ***String*** - Logical port mode (onMaster, onSlave, onSystem)

— Input: `abstractionDefPortID` of type ***String*** - Handle to abstractionDefPort element

F.7.5.6 getAbstractionDefPortIsAddress

Description: Returns isAddress for the given abstractionDefPort element

— Returns: `value` of type ***Boolean*** - Logical port isAddress

— Input: `abstractionDefPortID` of type ***String*** - Handle to abstractionDefPort element

F.7.5.7 getAbstractionDefPortIsData

Description: Returns isData for the given abstractionDefPort element

— Returns: `value` of type ***Boolean*** - Logical port isData

— Input: `abstractionDefPortID` of type ***String*** - Handle to abstractionDefPort element

F.7.5.8 getAbstractionDefPortIsClock

Description: Returns isClock for the given abstractionDefPort element

— Returns: `value` of type ***Boolean*** - Logical port isClock

— Input: `abstractionDefPortID` of type ***String*** - Handle to abstractionDefPort element

F.7.5.9 getAbstractionDefPortIsReset

Description: Returns isReset for the given abstractionDefPort element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

- Returns: `value` of type ***Boolean*** - Logical port isReset
- Input: `abstractionDefPortID` of type ***String*** - Handle to abstractionDefPort element

F.7.5.10 getAbstractionDefPortModeIDs

Description: Returns abtractionDefPortModeIDs for the given abstractionDefPort element

- Returns: `abstractionDefPortModeIDs` of type ***String[]*** - List of abstractionDefPortMode handles
- Input: `abstractionDefPortID` of type ***String*** - Handle to abstractionDefPort element

F.7.5.11 getAbstractionDefPortModeGroup

Description: Returns group for the given abstractionDefPortMode element

- Returns: `group` of type ***String*** - Logical port group
- Input: `abstractionDefPortModeID` of type ***String*** - Handle to abstractionDefPortMode element

F.7.5.12 getAbstractionDefPortModePresence

Description: Returns presence for the given abstractionDefPortMode element

- Returns: `presence` of type ***String*** - Logical port presence
- Input: `abstractionDefPortModeID` of type ***String*** - Handle to abstractionDefPortMode element

F.7.5.13 getAbstractionDefPortModeWidth

Description: Returns width for the given abstractionDefPortMode element

- Returns: `width` of type ***Number*** - Logical port width
- Input: `abstractionDefPortModeID` of type ***String*** - Handle to abstractionDefPortMode element

F.7.5.14 getAbstractionDefPortModeDirection

Description: Returns direction for the given abstractionDefPortMode element

- Returns: `direction` of type ***String*** - Logical port direction
- Input: `abstractionDefPortModeID` of type ***String*** - Handle to abstractionDefPortMode element

F.7.5.15 getAbstractionDefPortModeConstraintIDs

Description: Returns abstractionDefPortModeConstraintIDs for the given abstractionDefPortMode element

- Returns: `abstractionDefPortModeConstraintIDs` of type ***String[]*** - List of abstractionDefPortModeConstraint handles
- Input: `abstractionDefPortModeID` of type ***String*** - Handle to abstractionDefPortMode element

F.7.5.16 getAbstractionDefPortModeDriveConstraintIDs

Description: Returns driveContraintIDs for the given abstractionDefPortModeConstraint element

- Returns: `driveContraintIDs` of type ***String[]*** - DriveConstraint type

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

— Input: `abstractionDefPortModeConstraintID` of type ***String*** - Handle to abstractionDefPortModeConstraint element

F.7.5.17 getAbstractionDefPortModeLoadConstraintIDs

Description: Returns loadConstraintIDs for the given abstractionDefPortModeConstraint element

— Returns: `loadConstraintIDs` of type ***String[]*** - LoadConstraint type

— Input: `abstractionDefPortModeConstraintID` of type ***String*** - Handle to abstractionDefPortModeConstraint element

F.7.5.18 getAbstractionDefPortModeTimingConstraintIDs

Description: Returns timingContraintIDs for the given abstractionDefPortModeConstraint element

— Returns: `timingContraintIDs` of type ***String[]*** - TimingContraint clock name

— Input: `abstractionDefPortModeConstraintID` of type ***String*** - Handle to abstractionDefPortModeConstraint element

F.7.5.19 getAbstractionDefPortMirroredModeConstraintIDs

Description: Returns mirrored abstractionDefPortModeConstraintIDs for the given abstractionDefPortMode element

— Returns: `abstractionDefPortModeConstraintIDs` of type ***String[]*** - List of abstractionDefPortModeConstraint handles

— Input: `abstractionDefPortModeID` of type ***String*** - Handle to abstractionDefPortMode element

F.7.5.20 getAbstractionDefPortDefaultValue

Description: Returns defaultValue for the given abstractionDefPort element

— Returns: `value` of type ***Number*** - Logical port default value

— Input: `abstractionDefPortID` of type ***String*** - Handle to abstractionDefPort element

F.7.5.21 getAbstractionDefPortRequiresDriver

Description: Returns requiresDriver for the given abstractionDefPort element

— Returns: `value` of type ***Boolean*** - Logical port requiresDriver value

— Input: `abstractionDefPortID` of type ***String*** - Handle to abstractionDefPort element

F.7.5.22 getAbstractionDefPortRequiresDriverType

Description: Returns requiresDriverType for the given abstractionDefPort element

— Returns: `type` of type ***String*** - Logical port requiresDriver type value

— Input: `abstractionDefPortID` of type ***String*** - Handle to abstractionDefPort element

F.7.5.23 getAbstractionDefPortModeInitiative

Description: Returns initiative for the given abstractionDefPortMode element

— Returns: `initiative` of type ***String*** - Logical port initiative

— Input: `abstractionDefPortModeID` of type ***String*** - Handle to abstractionDefPortMode element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.5.24 getAbstractionDefPortModeKind

Description: Returns kind for the given abstractionDefPortMode element

- Returns: `kind` of type ***String*** - Logical port kind
- Input: `abstractionDefPortModeID` of type ***String*** - Handle to abstractionDefPortMode element

F.7.5.25 getAbstractionDefPortModeKindCustom

Description: Returns kind custom for the given abstractionDefPortMode element

- Returns: `kindCustom` of type ***String*** - Logical port kind custom
- Input: `abstractionDefPortModeID` of type ***String*** - Handle to abstractionDefPortMode element

F.7.5.26 getAbstractionDefPortModeBusWidth

Description: Returns busWidth for the given abstractionDefPortMode element

- Returns: `busWidth` of type ***Number*** - Logical port busWidth
- Input: `abstractionDefPortModeID` of type ***String*** - Handle to abstractionDefPortMode element

F.7.5.27 getAbstractionDefPortModeProtocolType

Description: Returns protocolType for the given abstractionDefPortMode element

- Returns: `type` of type ***String*** - Logical port type
- Input: `abstractionDefPortModeID` of type ***String*** - Handle to abstractionDefPortMode element

F.7.5.28 getAbstractionDefPortModeProtocolTypeCustom

Description: Returns protocolType custom for the given abstractionDefPortMode element

- Returns: `typeCustom` of type ***String*** - Logical port type custom
- Input: `abstractionDefPortModeID` of type ***String*** - Handle to abstractionDefPortMode element

F.7.5.29 getAbstractionDefPortModeProtocolPayloadName

Description: Returns payloadName for the given abstractionDefPortMode element

- Returns: `name` of type ***String*** - Logical port payload name
- Input: `abstractionDefPortModeID` of type ***String*** - Handle to abstractionDefPortMode element

F.7.5.30 getAbstractionDefPortModeProtocolPayloadType

Description: Returns payloadType for the given abstractionDefPortMode element

- Returns: `type` of type ***String*** - Logical port payload type
- Input: `abstractionDefPortModeID` of type ***String*** - Handle to abstractionDefPortMode element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.5.31 getAbstractionDefPortModeProtocolPayloadExtension

Description: Returns payloadExtension for the given abstractionDefPortMode element

- Returns: `extension` of type ***String*** - Logical port extension
- Input: `abstractionDefPortModeID` of type ***String*** - Handle to abstractionDefPortMode element

F.7.5.32 getAbstractionDefPortModeProtocolPayloadExtensionMandatory

Description: Returns payloadExtension mandatory for the given abstractionDefPortMode element

- Returns: `value` of type ***Boolean*** - Logical port extension mandatory
- Input: `abstractionDefPortModeID` of type ***String*** - Handle to abstractionDefPortMode element

F.7.6 Abstraction definition operations (EXTENDED)

F.7.6.1 createAbstractionDefinition

Description: Create a new abstractionDef with the given VLNV, busDefVLNV, logicalName and type and returns its abstractionDefID; fails and returns null if VLNV already exists

- Returns: `abstractionDefID` of type ***String*** - Handle to a new abstractionDef element
- Input: `abstractionDefVLNV` of type ***String[]*** - AbstractionDef VLNV
- Input: `busDefVLNV` of type ***String[]*** - BusDef VLNV
- Input: `logicalName` of type ***String*** - Logical port name
- Input: `type` of type ***String*** - Logical port style (wire or transactional)

F.7.6.2 setAbstractionDefinitionExtends

Description: Set extends with the given value for the given abstractionDef element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `abstractionDefID` of type ***String*** - handle to an abstractionDef element
- Input: `abstractionDefVLNV` of type ***String[]*** - AbstractionDef VLNV

F.7.6.3 addAbstractionDefinitionPort

Description: Add abstractionDefPort with the given logicalName and type to the given abstractionDef element

- Returns: `abstractionDefPortID` of type ***String*** - Handle to a new abstractionDefPort element
- Input: `abstractionDefID` of type ***String*** - handle to an abstractionDef element
- Input: `logicalName` of type ***String*** - Logical port name
- Input: `type` of type ***String*** - Logical port style (wire or transactional)

F.7.6.4 removeAbstractionDefinitionPort

Description: Remove the given abstractionDefPort element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `abstractionDefPortID` of type ***String*** - Handle to abstractionDefPort element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.6.5 setAbstractionDefPortAddress

Description: Set isAddress with the given value for the given abstractionDefPort element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `abstractionDefPortID` of type ***String*** - Handle to abstractionDefPort element
- Input: `isAddress` of type ***Boolean*** - Represents logical port an address

F.7.6.6 setAbstractionDefPortData

Description: Set isData with the given value for the given abstractionDefPort element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `abstractionDefPortID` of type ***String*** - Handle to abstractionDefPort element
- Input: `isData` of type ***Boolean*** - Represents logical port data

F.7.6.7 setAbstractionDefPortClock

Description: Set isClock with the given value for the given abstractionDefPort element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `abstractionDefPortID` of type ***String*** - Handle to abstractionDefPort element
- Input: `isClock` of type ***Boolean*** - Represents logical port a clock

F.7.6.8 setAbstractionDefPortReset

Description: Set isReset with the given value for the given abstractionDefPort element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `abstractionDefPortID` of type ***String*** - Handle to abstractionDefPort element
- Input: `isReset` of type ***Boolean*** - Represents logical port a reset

F.7.6.9 addAbstractionDefPortMode

Description: Add abstractionDefPortMode with the given value for the given abstractionDefPort element

- Returns: `abstractionDefPortModeID` of type ***String*** - Handle to a new abstractionDefPort-Mode element
- Input: `abstractionDefPortID` of type ***String*** - Handle to abstractionDefPort element
- Input: `mode` of type ***String*** - Logical port mode (onMaster, onSlave, onSystem)

F.7.6.10 setAbstractionDefPortModeGroup

Description: Set group with the given value for the given abstractionDefPortMode element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `abstractionDefPortModeID` of type ***String*** - Handle to abstractionDefPortMode element
- Input: `group` of type ***String*** - Logical port group

F.7.6.11 setAbstractionDefPortModePresence

Description: Set presence with the given value for the given abstractionDefPortMode element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

- Input: `abstractionDefPortModeID` of type ***String*** - Handle to abstractionDefPortMode element
- Input: `presence` of type ***String*** - Logical port presence

F.7.6.12 setAbstractionDefPortModeWidth

Description: Set width with the given value for the given abstractionDefPortMode element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `abstractionDefPortModeID` of type ***String*** - Handle to abstractionDefPortMode element
- Input: `width` of type ***String*** - Logical port width

F.7.6.13 setAbstractionDefPortModeDirection

Description: Set direction with the given value for the given abstractionDefPortMode element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `abstractionDefPortModeID` of type ***String*** - Handle to abstractionDefPortMode element
- Input: `direction` of type ***String*** - Logical port direction

F.7.6.14 addAbstractionDefPortModeDriveConstraint

Description: Add driveContraint with the given type for the given abstractionDefPortMode element

- Returns: `driveContraintID` of type ***String*** - Handle to a new driveContraint element
- Input: `abstractionDefPortModeID` of type ***String*** - Handle to abstractionDefPortMode element
- Input: `type` of type ***String*** - DriveConstraint type

F.7.6.15 addAbstractionDefPortModeLoadConstraint

Description: Add loadContraint with the given type for the given abstractionDefPortMode element

- Returns: `loadConstraintID` of type ***String*** - Handle to a new loadContraint element
- Input: `abstractionDefPortModeID` of type ***String*** - Handle to abstractionDefPortMode element
- Input: `type` of type ***String*** - LoadConstraint type

F.7.6.16 addAbstractionDefPortModeTimingConstraint

Description: Add timingContraint with the given type for the given abstractionDefPortMode element

- Returns: `timingContraintID` of type ***String*** - Handle to a new timingContraint element
- Input: `abstractionDefPortModeID` of type ***String*** - Handle to abstractionDefPortMode element
- Input: `clockName` of type ***String*** - TimingContraint clock name

F.7.6.17 addAbstractionDefPortMirroredModeDriveConstraint

Description: Add mirrored driveContraint with the given type for the given abstractionDefPortMode element

- Returns: `driveContraintID` of type ***String*** - Handle to a new driveContraint element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

— Input: `abstractionDefPortModeID` of type ***String*** - Handle to abstractionDefPortMode element
— Input: `type` of type ***String*** - DriveConstraint type

F.7.6.18 addAbstractionDefPortMirroredModeLoadConstraint

Description: Add mirrored loadContraint with the given type for the given abstractionDefPortMode element

— Returns: `loadConstraintID` of type ***String*** - Handle to a new loadContraint element
— Input: `abstractionDefPortModeID` of type ***String*** - Handle to abstractionDefPortMode element
— Input: `type` of type ***String*** - LoadConstraint type

F.7.6.19 addAbstractionDefPortMirroredModeTimingConstraint

Description: Add mirrored timingContraint with the given type for the given abstractionDefPortMode element

— Returns: `timingContraintID` of type ***String*** - Handle to a new timingContraint element
— Input: `abstractionDefPortModeID` of type ***String*** - Handle to abstractionDefPortMode element
— Input: `clockName` of type ***String*** - TimingContraint clock name

F.7.6.20 setAbstractionDefPortDefaultValue

Description: Set defaultValue with the given value for the given abstractionDefPort element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `abstractionDefPortID` of type ***String*** - Handle to abstractionDefPort element
— Input: `value` of type ***Number*** - Logical port default value

F.7.6.21 setAbstractionDefPortRequiresDriver

Description: Set requiresDriver with the given value for the given abstractionDefPort element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `abstractionDefPortID` of type ***String*** - Handle to abstractionDefPort element
— Input: `value` of type ***Boolean*** - Logical port requiresDriver value

F.7.6.22 setAbstractionDefPortRequiresDriverType

Description: Set driverType attribute of requiresDriver with the given value for the given abstractionDefPort element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `abstractionDefPortID` of type ***String*** - Handle to abstractionDefPort element
— Input: `type` of type ***String*** - Logical port requiresDriver type value

F.7.6.23 setAbstractionDefPortModeInitiative

Description: Set initiative with the given value for the given abstractionDefPort element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `abstractionDefPortID` of type ***String*** - Handle to abstractionDefPort element
— Input: `initiative` of type ***String*** - Logical port initiative

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.6.24 setAbstractionDefPortModeKind

Description: Set kind with the given value for the given abstractionDefPort element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `abstractionDefPortID` of type ***String*** - Handle to abstractionDefPort element
- Input: `kind` of type ***String*** - Logical port kind

F.7.6.25 setAbstractionDefPortModeKindCustom

Description: Set custom attribute of kind with the given value for the given abstractionDefPort element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `abstractionDefPortID` of type ***String*** - Handle to abstractionDefPort element
- Input: `kindCustom` of type ***String*** - Logical port kind custom

F.7.6.26 setAbstractionDefPortModeBusWidth

Description: Set busWidth with the given value for the given abstractionDefPort element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `abstractionDefPortID` of type ***String*** - Handle to abstractionDefPort element
- Input: `busWidth` of type ***Number*** - Logical port busWidth

F.7.6.27 setAbstractionDefPortModeProtocolType

Description: Set protocolType with the given value for the given abstractionDefPort element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `abstractionDefPortID` of type ***String*** - Handle to abstractionDefPort element
- Input: `type` of type ***String*** - Logical port type

F.7.6.28 setAbstractionDefPortModeProtocolTypeCustom

Description: Set custom attribute of protocolType with the given value for the given abstractionDefPort element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `abstractionDefPortID` of type ***String*** - Handle to abstractionDefPort element
- Input: `typeCustom` of type ***String*** - Logical port type custom

F.7.6.29 setAbstractionDefPortModeProtocolPayloadName

Description: Set payloadName with the given value for the given abstractionDefPort element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `abstractionDefPortID` of type ***String*** - Handle to abstractionDefPort element
- Input: `name` of type ***String*** - Logical port payload name

F.7.6.30 setAbstractionDefPortModeProtocolPayloadType

Description: Set payloadType with the given value for the given abstractionDefPort element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `abstractionDefPortID` of type ***String*** - Handle to abstractionDefPort element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

— Input: `type` of type ***String*** - Logical port payload type

F.7.6.31 setAbstractionDefPortModeProtocolPayloadExtension

Description: Set payloadExtension with the given value for the given abstractionDefPort element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

— Input: `abstractionDefPortID` of type ***String*** - Handle to abstractionDefPort element

— Input: `extension` of type ***String*** - Logical port extension

F.7.6.32 setAbstractionDefPortModeProtocolPayloadExtensionMandatory

Description: Set mandatory attribute of payloadExtension with the given value for the given abstractionDefPort element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

— Input: `abstractionDefPortID` of type ***String*** - Handle to abstractionDefPort element

— Input: `value` of type ***Boolean*** - Logical port extension mandatory

F.7.7 Abstractor operations (BASE)

F.7.7.1 getAbstractorAbstractorMode

Description: Returns the abstractorMode for the given abstractor or abstractorInstance element

— Returns: `mode` of type ***String*** - Abstractor mode value

— Input: `abstractorID` | `abstractorInstanceID` of type ***String*** - Handle to an abstractor or abstractorInstance element

F.7.7.2 getAbstractorBusType

Description: Returns the busType for the given abstractor or abstractorInstance element

— Returns: `busDefVLNV` of type ***String[]*** - BusDef VLNV

— Input: `abstractorID` | `abstractorInstanceID` of type ***String*** - Handle to an abstractor or abstractorInstance element

F.7.7.3 getAbstractorAbstractorInterfaceIDs

Description: Returns the abstractorInterfaceIDs for the given abstractor or abstractorInstance element

— Returns: `abstractorBusInterfaceIDs` of type ***String[]*** - List of abstractorBusInterface handles

— Input: `abstractorID` | `abstractorInstanceID` of type ***String*** - Handle to an abstractor or abstractorInstance element

F.7.7.4 getAbstractorBusInterfaceAbstractionTypeIDs

Description: Returns the abstractionTypeIDs for the given abstractorBusInterface element

— Returns: `abstractionTypeIDs` of type ***String[]*** - List of abstractionType handles

— Input: `abstractorBusInterfaceID` of type ***String*** - Handle to an abstractorBusInterface element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.7.5 getAbstractorViewIDs

Description: Returns the abstractorViewIDs for the given abstractor or abstractorInstance element

- Returns: `abstractorViewIDs` of type ***String[]*** - List of abstractorView handles
- Input: `abstractorID` | `abstractorInstanceID` of type ***String*** - Handle to an abstractor or abstractorInstance element

F.7.7.6 getAbstractorComponentInstantiationsID

Description: Returns the componentInstantationIDs for the given abstractor or abstractorInstance element

- Returns: `componentInstantiationIDs` of type ***String[]*** - List of componentInstantiation handles
- Input: `abstractorID` | `abstractorInstanceID` of type ***String*** - Handle to an abstractor or abstractorInstance element

F.7.7.7 getAbstractorPortIDs

Description: Returns the portIDs for the given abstractor or abstractorInstance element

- Returns: `portIDs` of type ***String[]*** - List of port handles
- Input: `abstractorID` | `abstractorInstanceID` of type ***String*** - Handle to an abstractor or abstractorInstance element

F.7.7.8 getAbstractorGeneratorIDs

Description: Returns the generatorIDs for the given abstractor or abstractorInstance element

- Returns: `generatorIDs` of type ***String[]*** - List of generator handles
- Input: `abstractorID` | `abstractorInstanceID` of type ***String*** - Handle to an abstractor or abstractorInstance element

F.7.7.9 getAbstractorChoiceIDs

Description: Returns the choiceIDs for the given abstractor or abstractorInstance element

- Returns: `choiceIDs` of type ***String[]*** - List of choice handles
- Input: `abstractorID` | `abstractorInstanceID` of type ***String*** - Handle to an abstractor or abstractorInstance element

F.7.7.10 getAbstractorFileSetIDs

Description: Returns the fileSetIDs for the given abstractor or abstractorInstance element

- Returns: `fileSetIDs` of type ***String[]*** - List of fileSet handles
- Input: `abstractorID` | `abstractorInstanceID` of type ***String*** - Handle to an abstractor or abstractorInstance element

F.7.8 Abstractor operations (EXTENDED)

F.7.8.1 createAbstractor

Description: Create a new abstractor with the given VLNV and returns its abstractorID; fails and returns null if VLNV already exists

- Returns: `abstractorID` of type ***String*** - Handle to a new abstractor

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

- Input: `abstractorVLNV` of type ***String[]*** - VLNV for a new abstractor
- Input: `mode` of type ***String*** - Abstractor mode
- Input: `busDefVLNV` of type ***String*** - BusDef VLNV
- Input: `busInterfaceName` of type ***String*** - First busInterface name
- Input: `busInterfaceName` of type ***String*** - Second busInterfaceName

F.7.8.2 addAbstractorBusInterfaceAbstractionType

Description: Add an abstractionType with the given abstractionRef for the given abstractorBusInterface element

- Returns: `abstractionTypeID` of type ***String*** - Handle to a new abstractionType
- Input: `abstractorBusInterfaceID` of type ***String*** - Handle to an abstractorBusInterface element
- Input: `abstractionRef` of type ***String[]*** - AbstractionDef VLNV

F.7.8.3 removeAbstractorBusInterfaceAbstractionType

Description: Remove the given abstractionType element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `abstractionTypeID` of type ***String*** - Handle to an abstractionType element

F.7.8.4 addAbstractorView

Description: Add an abstractorView with the given name and envIdentifier for the given abstractor element

- Returns: `abstractorViewID` of type ***String*** - Handle to a new abstractorView
- Input: `abstractorID` of type ***String*** - Handle to an abstractor element
- Input: `name` of type ***String*** - Abstractor view name
- Input: `envIdentifier` of type ***String[]*** - Abstractor view envIdentifier

F.7.8.5 removeAbstractorView

Description: Remove the given abstractorView element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `abstractorViewID` of type ***String*** - Handle to an abstractorView element

F.7.8.6 addAbstractorComponentInstantiation

Description: Add a componentInstantiation with the given name for the given abstractor element

- Returns: `componentInstantiationID` of type ***String*** - Handle to a new componentInstantiation
- Input: `abstractorID` of type ***String*** - Handle to an abstractor element
- Input: `name` of type ***String*** - ComponentIntstantiation name

F.7.8.7 removeAbstractorComponentInstantiation

Description: Remove the given componentInstantiation element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `componentInstantiationID` of type ***String*** - Handle to a componentInstantiation element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.8.8 addAbstractorPort

Description: Add a port with the given name, type, and directionOrInitiative for the given abstractor element

- Returns: `portID` of type ***String*** - Handle to a new port
- Input: `abstractorID` of type ***String*** - Handle to an abstractor element
- Input: `name` of type ***String*** - port name
- Input: `type` of type ***String*** - Port type (wire or transactional)
- Input: `directionOrInitiative` of type ***String*** - Port direction or initiative

F.7.8.9 removeAbstractorPort

Description: Remove the given port element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `portID` of type ***String*** - Handle to a port element

F.7.8.10 addAbstractorGenerator

Description: Add a generator with the given name and path to the given abstractor element

- Returns: `generatorID` of type ***String*** - Handle to a new generator
- Input: `abstractorID` of type ***String*** - Handle to an abstractor element
- Input: `name` of type ***String*** - Generator name
- Input: `generatorExecutable` of type ***String*** - Path to generator executable

F.7.8.11 removeAbstractorGenerator

Description: Remove the given generator element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `generatorID` of type ***String*** - Handle to a generator element

F.7.8.12 addAbstractorChoice

Description: Add a choice with the given name and enumerations to the given abstractor element

- Returns: `choiceID` of type ***String*** - Handle to a new choice
- Input: `abstractorID` of type ***String*** - Handle to an abstractor element
- Input: `name` of type ***String*** - Choice name
- Input: `enumerations` of type ***String[]*** - List of enumerations

F.7.8.13 removeAbstractorChoice

Description: Remove the given choice element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `choiceID` of type ***String*** - Handle to a choice element

F.7.8.14 addAbstractorFileSet

Description: Add a fileSet with the given name to the given abstractor element

- Returns: `fileSetID` of type ***String*** - Handle to a new fileSet
- Input: `abstractorID` of type ***String*** - Handle to an abstractor element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

— Input: `name` of type ***String*** - FileSet name

F.7.8.15 removeAbstractorFileSet

Description: Remove the given fileSet element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

— Input: `fileSetID` of type ***String*** - Handle to a fileSet element

F.7.9 Abstractor view operations (BASE)

F.7.9.1 getAbstractorViewEnvIdentifier

Description: Returns envIdentifier for the given abstractorView element

— Returns: `envIdentifier` of type ***String[]*** - View envIdentifier

— Input: `abstractorViewID` of type ***String*** - Handle to an abstractorView element

F.7.9.2 getAbstractorViewComponentInstantiationRef

Description: Returns componentInstantiationRef for the given abstractorView element

— Returns: `componentInstantiationName` of type ***String*** - Name of componentInstantiation

— Input: `abstractorViewID` of type ***String*** - Handle to an abstractorView element

F.7.9.3 getAbstractorViewComponentInstantiationRefID

Description: Returns componentInstantiationID for the given abstractorView element

— Returns: `componentInstantiationID` of type ***String*** - Handle to a componentInstantiation

— Input: `abstractorViewID` of type ***String*** - Handle to an abstractorView element

F.7.10 Abstractor view operations (EXTENDED)

F.7.10.1 setAbstractorViewComponentInstantiationRef

Description: Set componentInstantiationRef for the given abstractorView element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

— Input: `abstractorViewID` of type ***String*** - Handle to an abstractorView element

— Input: `componentInstantiationRef` of type ***String*** - componentInstantiation name

F.7.11 Access handle operations (BASE)

F.7.11.1 getAccessHandleForce

Description: Returns the value of attribute force for a given accessHandle element

— Returns: `force` of type ***Boolean*** - Value of the force attribute

— Input: `accessHandleID` of type ***String*** - Handle to an accessHandle element

F.7.11.2 getAccessHandleViewIDs

Description: Returns the viewIDs for a given accessHandle element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

- Returns: `viewIDs` of type ***String[]*** - List of view handles
- Input: `accessHandleID` of type ***String*** - Handle to an accessHandle element

F.7.11.3 getAccessHandleIndices

Description: Returns the indices for a given accessHandle element

- Returns: `indices` of type ***Number[]*** - List of indices
- Input: `accessHandleID` of type ***String*** - Handle to an accessHandle element

F.7.11.4 getAccessHandleSliceIDs

Description: Returns the slideIDs for a given accessHandle element

- Returns: `sliceIDs` of type ***String[]*** - List of slice handles
- Input: `accessHandleID` of type ***String*** - Handle to an accessHandle element

F.7.12 Access handle operations (EXTENDED)

F.7.12.1 setAccessHandleForce

Description: Set the given value for the attribute force for the given accessHandle element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `accessHandleID` of type ***String*** - Handle to an accessHandle element
- Input: `force` of type ***Boolean*** - Value of the attribute force

F.7.12.2 addAccessHandleViewName

Description: Add a viewRef with the given name to the given accessHandle element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `accessHandleID` of type ***String*** - Handle to an accessHandle element
- Input: `viewName` of type ***String*** - View name

F.7.12.3 removeAccessHandleViewName

Description: Remove the viewRef with the given name from the given accessHandle element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `accessHandleID` of type ***String*** - Handle to an accessHandle element
- Input: `viewName` of type ***String*** - View name

F.7.12.4 addAccessHandleSlice

Description: Add a slice with the given pathSegmentNames and the given indices to the given accessHandle element

- Returns: `sliceID` of type ***String*** - Handle to a new slice
- Input: `accessHandleID` of type ***String*** - Handle to an accessHandle element
- Input: `pathSegmentNames` of type ***String[]*** - List of pathSegmentNames
- Input: `indices` of type ***String[]*** - List of indices

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.12.5 removeAccessHandleSlice

Description: Remove the given slice

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `sliceID` of type ***String*** - Handle to a slice element

F.7.13 Address space operations (BASE)

F.7.13.1 getAddressSpaceRange

Description: Returns the range of the given addressSpace element

- Returns: `range` of type ***Number*** - AddressSpace range
- Input: `addressSpaceID` of type ***String*** - Handle to an addressSpace element

F.7.13.2 getAddressSpaceWidth

Description: Returns the width of the given addressSpace element

- Returns: `width` of type ***Number*** - AddressSpace width
- Input: `addressSpaceID` of type ***String*** - Handle to an addressSpace element

F.7.13.3 getAddressSpaceSegmentIDs

Description: Returns the segmentIDs of the given addressSpace element

- Returns: `segmentIDs` of type ***String[]*** - List of segment handles
- Input: `addressSpaceID` of type ***String*** - Handle to an addressSpace element

F.7.13.4 getSegmentAddressOffset

Description: Returns the addressOffset of the given segment element

- Returns: `addressOffset` of type ***Number*** - Segment address offset
- Input: `segmentID` of type ***String*** - Handle to a segment element

F.7.13.5 getSegmentRange

Description: Returns the range of the given segment element

- Returns: `range` of type ***Number*** - Segment range
- Input: `segmentID` of type ***String*** - Handle to a segment element

F.7.13.6 getAddressSpaceAddressUnitBits

Description: Returns the addressUnitBits for the given addressSpace element

- Returns: `addressUnitBits` of type ***Number*** - AddressSpace addressUnitBits
- Input: `addressSpaceID` of type ***String*** - Handle to an addressSpace element

F.7.13.7 getAddressSpaceExecutableImageIDs

Description: Returns the executableImageIDs for the given addressSpace element

- Returns: `executableImageIDs` of type ***String[]*** - List of executableImage handles
- Input: `addressSpaceID` of type ***String*** - Handle to an addressSpace element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.13.8 getExecutableImageFileBuilderIDs

Description: Returns the fileBuilderIds for the given executableImage element

- Returns: `fileBuilderIDs` of type ***String[]*** - List of fileBuilder handles
- Input: `executableImageID` of type ***String*** - Handle to an executableImage element

F.7.13.9 getFileBuilderFileType

Description: Returns the fileType for the given fileBuilder element

- Returns: `fileType` of type ***String*** - FileBuilder FileType
- Input: `fileBuilderID` of type ***String*** - Handle to a fileBuilder element

F.7.13.10 getFileBuilderCommand

Description: Returns the command for the given fileBuilder element

- Returns: `command` of type ***String*** - FileBuilder command
- Input: `fileBuilderID` of type ***String*** - Handle to a fileBuilder element

F.7.13.11 getFileBuilderFlags

Description: Returns the flags for the given fileBuilder element

- Returns: `flags` of type ***String*** - FileBuilder flags
- Input: `fileBuilderID` of type ***String*** - Handle to a fileBuilder element

F.7.13.12 getFileBuilderReplaceDefaultFlags

Description: Returns the replaceDefaultFlags for the given fileBuilder element

- Returns: `value` of type ***Boolean*** - FileBuilder replaceDefaultFlags
- Input: `fileBuilderID` of type ***String*** - Handle to a fileBuilder element

F.7.13.13 getExecutableImageLinker

Description: Returns the linker for the given executableImage element

- Returns: `linker` of type ***String*** - Linker value
- Input: `executableImageID` of type ***String*** - Handle to an executableImage element

F.7.13.14 getExecutableImageLinkerFlags

Description: Returns the linkerFlags for the given executableImage element

- Returns: `linkerFlags` of type ***String*** - LinkerFlags value
- Input: `executableImageID` of type ***String*** - Handle to an executableImage element

F.7.13.15 getExecutableImageLinkerCommandFileID

Description: Returns the linkerCommandFileID for the given executableImage element

- Returns: `linkerCommandFileID` of type ***String*** - Handle to a linkerCommandFile
- Input: `executableImageID` of type ***String*** - Handle to an executableImage element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.13.16 getLinkerCommandFileName

Description: Returns the name for a given linkerCommandFile

- Returns: `fileName` of type ***Uri*** - LinkerCommandFile filename
- Input: `linkerCommandFileID` of type ***String*** - Handle to a linkerCommandFile

F.7.13.17 getLinkerCommandFileLineSwitch

Description: Returns the commandLineSwitch for a given linkerCommandFile

- Returns: `value` of type ***String*** - LinkerCommandFile commandLineSwitch value
- Input: `linkerCommandFileID` of type ***String*** - Handle to a linkerCommandFile

F.7.13.18 getLinkerCommandFileEnable

Description: Returns the enable for a given linkerCommandFile

- Returns: `value` of type ***Boolean*** - LinkerCommandFile enable value
- Input: `linkerCommandFileID` of type ***String*** - Handle to a linkerCommandFile

F.7.13.19 getLinkerCommandGeneratorIDs

Description: Returns the generatorIDs for a given linkerCommandFile

- Returns: `generatorIDs` of type ***String[]*** - List of generator handles
- Input: `linkerCommandFileID` of type ***String*** - Handle to a linkerCommandFile

F.7.13.20 getAddressSpaceLocalMemoryMapID

Description: Returns the localMemoryMapID for a given addressSpace element

- Returns: `localMemoryMapID` of type ***String*** - Handle to a localMemoryMap
- Input: `addressSpaceID` of type ***String*** - Handle to an addressSpace element

F.7.13.21 getLocalMemoryMapAddressBlockIDs

Description: Returns the localAddressBlockIDs for a given localMemory map element

- Returns: `localAddressBlockIDs` of type ***String[]*** - Handle to a localAddressBlock
- Input: `addressSpaceID` of type ***String*** - Handle to an addressSpace element

F.7.13.22 getLocalMemoryMapBankIDs

Description: Returns the localBankIDs for a given localMemory map element

- Returns: `localBankIDs` of type ***String[]*** - Handle to a localBank
- Input: `addressSpaceID` of type ***String*** - Handle to an addressSpace element

F.7.14 Address space operations (EXTENDED)

F.7.14.1 addAddressSpaceSegment

Description: Add a segment with the given name, addressOffset, and range to the given addressSpace element

- Returns: `segmentID` of type ***String*** - Handle to a new segment element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

- Input: `addressSpaceID` of type ***String*** - Handle to an addressSpace element
- Input: `name` of type ***String*** - Segment name
- Input: `addressOffset` of type ***String*** - Segment addressOffset
- Input: `range` of type ***String*** - Segment range

F.7.14.2 removeAddressSpaceSegment

Description: Remove the given segment element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `segmentID` of type ***String*** - Handle to a segment element

F.7.14.3 setAddressSpaceAddressUnitBits

Description: Set the addressUnitBits with the given value to the given address space element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `addressSpaceID` of type ***String*** - Handle to an addressSpace element
- Input: `addressUnitBits` of type ***Number*** - AddressSpace addressUnitBits

F.7.14.4 addAddressSpaceExecutableImage

Description: Add an executableImage with the given name to the given address space element

- Returns: `executableImageID` of type ***String*** - Handle to a new executableImage element
- Input: `addressSpaceID` of type ***String*** - Handle to an addressSpace element
- Input: `name` of type ***String*** - ExecutableImage name

F.7.14.5 removeAddressSpaceExecutableImage

Description: Remove the given executableImage element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `executableImageID` of type ***String*** - Handle to an executableImage element

F.7.14.6 addExecutableImageFileBuilderID

Description: Add a fileBuilder with the given fileType and command to the given executableImage element

- Returns: `fileBuilderID` of type ***String*** - Handle to a new fileBuilder element
- Input: `executableImageID` of type ***String*** - Handle to an executableImage element
- Input: `fileType` of type ***String*** - FileBuilder fileType
- Input: `command` of type ***String*** - FileBuilder commands

F.7.14.7 removeExecutableImageFileBuilderID

Description: Remove the given fileBuilder element

- Returns: `status` of type Boolean - Indicates call is successful (true) or not (false)
- Input: `fileBuilderID` of type ***String*** - Handle to a fileBuilder element

F.7.14.8 setFileBuilderFlags

Description: Set the flags with the given value for the given fileBuilder element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

- Input: `fileBuilderID` of type ***String*** - Handle to a fileBuilder element
- Input: `flags` of type ***String*** - FileBuilder flags

F.7.14.9 setFileBuilderReplaceDefaultFlags

Description: Set the replaceDefaultFlags with the given value for the given fileBuilder element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `fileBuilderID` of type ***String*** - Handle to a fileBuilder element
- Input: `value` of type ***Boolean*** - FileBuilder replaceDefaultFlags

F.7.14.10 setExecutableImageLinker

Description: Set the linker with the given value for the given executableImage element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `executableImageID` of type ***String*** - Handle to an executableImage element
- Input: `linker` of type ***String*** - ExecutableImage linker

F.7.14.11 setExecutableImageLinkerFlags

Description: Set the linkerFlags with the given value for the given executableImage element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `executableImageID` of type ***String*** - Handle to an executableImage element
- Input: `linkerFlags` of type ***String*** - ExecutableImage linkerFlags

F.7.14.12 addExecutableImageLinkerCommandFile

Description: Set the linkerCommandFile with the given value for the given executableImage element

- Returns: `linkerCommandFileID` of type ***String*** - Handle to a new linkerCommandFile element
- Input: `executableImageID` of type ***String*** - Handle to an executableImage element
- Input: `name` of type ***String*** - ExecutableImage linkerCommandFile
- Input: `commandLineSwitch` of type ***String*** - Flag on the command line specifying the linker command file
- Input: `enable` of type ***Boolean*** - Indicates whether to use this linker command file in the default scenario

F.7.14.13 removeExecutableImageLinkerCommandFile

Description: Remove the given linkerCommandFile element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `linkerCommandFileID` of type ***String*** - Handle to a linkerCommandFile

F.7.14.14 addLinkerCommandFileGenerator

Description: Set the generatorRef with the given value for the given linkerCommandFile element

- Returns: `generatorID` of type ***String*** - Handle to a new generator element
- Input: `linkerCommandFileID` of type ***String*** - Handle to a linkerCommandFile
- Input: `generatorRef` of type ***String*** - Generator name

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.14.15 removeLinkerCommandFileGenerator

Description: Remove the given generator element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `generatorID` of type ***String*** - Handle to a generator element

F.7.14.16 addLocalMemoryMapAddressBlock

Description: Add an addressBlock with the given name, baseAddress, range, and width to the given localMemoryMap element

- Returns: `addressBlockID` of type ***String*** - Handle to a new addressBlock element
- Input: `localMemoryMapID` of type ***String*** - Handle to a localMemoryMap element
- Input: `name` of type ***String*** - AddressBlock name
- Input: `baseAddress` of type ***Number*** - AddressBlock baseAddress
- Input: `range` of type ***Number*** - AddressBlock range
- Input: `width` of type ***Number*** - AddressBlock width

F.7.14.17 removeLocalMemoryMapAddressBlock

Description: Remove the given addressBlock element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `addressBlockID` of type ***String*** - Handle to an addressBlock element

F.7.14.18 addLocalMemoryMapBank

Description: Add a localBank with the given name and baseAddress to the given localMemoryMap element

- Returns: `localBankID` of type ***String*** - Handle to a new localBank element
- Input: `addressBlockID` of type ***String*** - Handle to an addressBlock element
- Input: `name` of type ***String*** - LocalBank name
- Input: `baseAddress` of type ***Number*** - LocalBank baseAddress

F.7.14.19 removeLocalMemoryMapBank

Description: Remove the given localBank element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `localBankID` of type ***String*** - Handle to a localBank element

F.7.15 Array operations (BASE)

F.7.15.1 getArrayIDs

Description: Returns arraryIDs of the given element

- Returns: `arrayIDs` of type ***String[]*** - List of array handles
- Input: `arrayContainerElementID` of type ***String*** - Handle to an element that has an array element

F.7.15.2 getArrayRange

Description: Returns range of the given array element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

— Returns: `range` of type ***Number[]*** - Range with left in index 0 and right in index 1
— Input: `arraryID` of type ***String*** - Handle to an array element

F.7.15.3 getArraryRangeExpression

Description: Returns the range expressions of the given array element

— Returns: `rangeExpression` of type ***String[]*** - Range expression with left index 0 and right in index 1
— Input: `arraryID` of type ***String*** - Handle to an array element

F.7.16 Array operations (EXTENDED)

F.7.16.1 addArray

Description: Add an array to the given element

— Returns: `arraryID` of type ***String*** - Handle to new array
— Input: `arrayContainerElementID` of type ***String*** - Handle to an element that has an array element
— Input: `range` of type ***String[]*** - Range expression with left in index 0 and right in index 1

F.7.16.2 removeArray

Description: Remove the given array

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `arraryID` of type ***String*** - Handle to an array element

F.7.17 Assertion operations (BASE)

F.7.17.1 getAssertionIDs

Description: Returns assertionIDs of the given element

— Returns: `assertionIDs` of type ***String[]*** - List of assertion handles
— Input: `assertionContainerElementID` of type ***String*** - Handle to an element that has assertion elements

F.7.17.2 getAssertionValue

Description: Returns value of the given assertion element

— Returns: `value` of type ***Boolean*** - Assertion value
— Input: `assertionID` of type ***String*** - Handle to an assertion element

F.7.17.3 getAssertionValueExpression

Description: Returns expression of the given assertion element

— Returns: `expression` of type ***String*** - Assertion expression
— Input: `assertionID` of type ***String*** - Handle to an assertion element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.17.4 addAssertion

Description: Add an assertion with the given name and given value to the given element

- Returns: `assertionID` of type ***String*** - Handle to a new assertion
- Input: `assertionContainerElementID` of type ***String*** - Handle to an element that has assertion elements
- Input: `name` of type ***String*** - Assertion name
- Input: `expression` of type ***String*** - Assertion expression

F.7.17.5 removeAssertion

Description: Remove the given assertion

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `assertionID` of type ***String*** - Handle to an assertion element

F.7.17.6 setAssertionValue

Description: Set the value of the given assertion

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `assertionID` of type ***String*** - Handle to an assertion element
- Input: `expression` of type ***String*** - Assertion expression

F.7.18 BitsInLau get operations (BASE)

F.7.18.1 getBitsInLauValue

Description: Returns bitsInLau value in the given element

- Returns: `value` of type ***Number*** - BitsInLau value
- Input: `elementID` of type ***String*** - Handle to an element that has a bitsInLau element

F.7.18.2 getBitsInLauValueExpression

Description: Returns bitsInLau expression in the given element

- Returns: `expression` of type ***String*** - BitsInLau expression
- Input: `elementID` of type ***String*** - Handle to an element that has a bitsInLau element

F.7.19 BitsInLau get operations (EXTENDED)

F.7.19.1 addBitsInLauValue

Description: Add the given bitsInLau expression to the given element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to an element that has a bitsInLau element
- Input: `expression` of type ***string*** - BitsInLau expression

F.7.19.2 removeBitsInLauValue

Description: Remove the bitsInLau element from the given element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `elementID` of type ***String*** - Handle to an element that has a bitsInLau element

F.7.19.3 setBitsInLauValue

Description: Set the given bitsInLau expression in the given element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `elementID` of type ***String*** - Handle to an element that has a bitsInLau element
— Input: `expression` of type ***String*** - BitsInLau expression

F.7.20 Bridge operations (BASE)

F.7.20.1 getBridgeMasterID

Description: Returns the busInterfaceID of the masterRef attribute in the given bridge element

— Returns: `busInterfaceID` of type ***String*** - Handle to bus interface master
— Input: `bridgeID` of type ***String*** - Handle to a bridge element

F.7.21 Bridge operations (EXTENDED)

F.7.21.1 setBridgeMasterID

Description: Set the value of the masterRef attribute for the given busInterfaceID in the given bridge element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `bridgeID` of type ***String*** - Handle to a bridge element
— Input: `busInterfaceID` of type ***String*** - Handle to a bus interface master

F.7.22 Bus definition operations (BASE)

F.7.22.1 getBusDefinitionDirectConnection

Description: Returns directConnection for a given busDef or busDefInstance element

— Returns: `value` of type ***Boolean*** - BusDef directionConnection
— Input: `busDefID` | `busDefInstanceID` of type ***String*** - Handle to a busDef or busDefInstance element

F.7.22.2 getBusDefinitionBroadcast

Description: Returns broadcast for a given busDef or busDefInstance element

— Returns: `value` of type ***Boolean*** - BusDef broadcast
— Input: `busDefID` | `busDefInstanceID` of type ***String*** - Handle to a busDef or busDefInstance element

F.7.22.3 getBusDefinitionIsAddressable

Description: Returns isAddressable for a given busDef or busDefInstance element

— Returns: `value` of type ***Boolean*** - BusDef isAddressable

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

- Input: `busDefID` | `busDefInstanceID` of type ***String*** - Handle to a busDef or busDefInstance element

F.7.22.4 getBusDefinitionExtends

Description: Returns extends for a given busDef or busDefInstance element

- Returns: `busDefVLNV` of type ***String[]*** - BusDef VLNV
- Input: `busDefID` | `busDefInstanceID` of type String - Handle to a busDef or busDefInstance element

F.7.22.5 getBusDefinitionMaxMasters

Description: Returns maxMasters for a given busDef or busDefInstance element

- Returns: `value` of type ***Number*** - BusDef maxMasters
- Input: `busDefID` | `busDefInstanceID` of type ***String*** - Handle to a busDef or busDefInstance element

F.7.22.6 getBusDefinitionMaxSlaves

Description: Returns maxSlaves for a given busDef or busDefInstance element

- Returns: `value` of type ***Number*** - BusDef maxSlaves
- Input: `busDefID` | `busDefInstanceID` of type ***String*** - Handle to a busDef or busDefInstance element

F.7.22.7 getBusDefinitionSystemGroupNames

Description: Returns systemGroupNames for a given busDef or busDefInstance element

- Returns: `groupNames` of type ***String[]*** - BusDef systemGroupNames
- Input: `busDefID` | `busDefInstanceID` of type ***String*** - Handle to a busDef or busDefInstance element

F.7.23 Bus definition operations (EXTENDED)

F.7.23.1 createBusDefinition

Description: Create a new busDef with the given VLNV, directConnection, and isAddressable and returns its busDefID; fails and returns null if VLNV already exists

- Returns: `busDefID` of type ***String*** - Handle to a new busDef element
- Input: `busDefVLNV` of type ***string[]*** - BusDef VLNV
- Input: `directConnection` of type ***Boolean*** - BusDef directionConnection
- Input: `isAddressable` of type ***Boolean*** - BusDef isAddressable

F.7.23.2 setBusDefinitionDirectConnection

Description: Set directionConnection with the given value for the given busDef element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `busDefID` of type ***String*** - Handle to a busDef element
- Input: `value` of type ***Boolean*** - BusDef directionConnection

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.23.3 setBusDefinitionBroadcast

Description: Set broadcast with the given value for the given busDef element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `busDefID` of type ***String*** - Handle to a busDef element
- Input: `value` of type ***Boolean*** - BusDef broadcast

F.7.23.4 setBusDefinitionIsAddressable

Description: Set isAddressable with the given value for the given busDef element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `busDefID` of type ***String*** - Handle to a busDef element
- Input: `value` of type ***Boolean*** - BusDef isAddressable

F.7.23.5 setBusDefinitionExtends

Description: Set extends with the given value for the given busDef element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `busDefID` of type ***String*** - Handle to a busDef element
- Input: `busDefVLNV` of type ***String[]*** - BusDef VLNV

F.7.23.6 setBusDefinitionMaxMasters

Description: Set maxMasters with the given value for the given busDef element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `busDefID` of type ***String*** - Handle to a busDef element
- Input: `value` of type ***Number*** - BusDef maxMasters

F.7.23.7 setBusDefinitionMaxSlaves

Description: Set maxSlaves with the given value for the given busDef element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `busDefID` of type ***String*** - Handle to a busDef element
- Input: `value` of type ***Number*** - BusDef maxSlaves

F.7.23.8 setBusDefinitionSystemGroupNames

Description: Set systemGroupNames with the given value for the given busDef element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `busDefID` of type ***String*** - Handle to a busDef element
- Input: `groupNames` of type ***String[]*** - BusDef systemGroupNames

F.7.24 Businterface operations (BASE)

F.7.24.1 getBusInterfaceBusInstanceID

Description: Returns the busDefInstanceID for a given busInterface element

- Returns: `busDefInstanceID` of type ***String*** - Handle to a busDefInstance

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

— Input: `busInterfaceID` of type ***String*** - Handle to a busInterface element

F.7.24.2 getBusInterfaceAbstractionTypeIDs

Description: Returns the abstractionTypeIDs for a given busInterface element

— Returns: `abstractionTypeIDs` of type ***String[]*** - List of abstractionType handles

— Input: `busInterfaceID` of type ***String*** - Handle to a busInterface element

F.7.24.3 getAbstractionTypeViewRefs

Description: Returns the viewRefs for a given abstractionType element

— Returns: `viewNames` of type ***String[]*** - List of view names

— Input: `abstractionTypeID` of type ***String*** - Handle to an abstractionType element

F.7.24.4 getAbstractionTypeAbstractionInstanceID

Description: Returns the abstractionDefInstanceID for a given abstractionType element

— Returns: `abstractionDefInstanceID` of type ***String*** - Handle to a abstractionDefInstance

— Input: `abstractionTypeID` of type ***String*** - Handle to an abstractionType element

F.7.24.5 getAbstractionTypePortMapIDs

Description: Returns the portMapIDs for a given abstractionType element

— Returns: `portMapIDs` of type ***String[]*** - List of portMap handles

— Input: `abstractionTypeID` of type ***String*** - Handle to an abstractionType element

F.7.24.6 getPortMapLogicalPortID

Description: Returns the logicalPortID for a given portMap element

— Returns: `logicalPortID` of type ***String*** - Handle to a logicalPort

— Input: `portMapID` of type ***String*** - Handle to a portMap element

F.7.24.7 getPortMapPhysicalPortID

Description: Returns the physicalPortID for a given portMap element

— Returns: `physicalPortID` of type ***String*** - Handle to a physicalPort

— Input: `portMapID` of type ***String*** - Handle to a portMap element

F.7.24.8 getPortMapRange

Description: Returns the range for a given logicalPort or physicalPort element

— Returns: `range` of type ***Number[]*** - Range with left at index 0 and right at index 1

— Input: `logicalPortID` | `physicalPortID` of type ***String*** - logicalPortID | physicalPortID

F.7.24.9 getPortMapIndices

Description: Returns the indices for a given physicalPort element

— Returns: `indices` of type ***Number[]*** - List of indices

— Input: `physicalPortID` of type ***String*** - Handle to a physicalPort element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.24.10 getPortMapLogicalTieOff

Description: Returns the logicalTieOff for a given portMap element

- Returns: `value` of type ***String*** - Tie off value for logical port in port map
- Input: `portMapID` of type ***String*** - Handle to a portMap element

F.7.24.11 getPortMapIsInformative

Description: Returns the isInformative for a given portMap element

- Returns: `value` of type ***Boolean*** - Indicates if portmap should be nestlisted (false) or not (true)
- Input: `portMapID` of type ***String*** - Handle to a portMap element

F.7.24.12 getBusInterfaceInterfaceMode

Description: Returns the interfaceMode for a given busInterface element

- Returns: `interfaceMode` of type ***String*** - Interface mode (master, slave, system, etc)
- Input: `busInterfaceID` of type ***String*** - Handle to a busInterface element

F.7.24.13 getBusInterfaceMasterAddressSpaceID

Description: Returns the addressSpaceID for a given busInterface element (master only)

- Returns: `addressSpaceID` of type ***String*** - Handle to an addressSpace
- Input: `busInterfaceID` of type ***String*** - Handle to a busInterface element

F.7.24.14 getBusInterfaceMasterBaseAddress

Description: Returns the baseAddress for a given busInterface element (master only)

- Returns: `baseAddress` of type ***Number*** - Base address
- Input: `busInterfaceID` of type ***String*** - Handle to a busInterface element

F.7.24.15 getBusInterfaceSlaveMemoryMapID

Description: Returns the memoryMapID for a given busInterface element (slave only)

- Returns: `memoryMapID` of type ***String*** - Handle to a memoryMap
- Input: `busInterfaceID` of type ***String*** - Handle to a busInterface element

F.7.24.16 getBusInterfaceSlaveBridgeIDs

Description: Returns the bridgeIDs for a given busInterface element (slave only)

- Returns: `bridgeIDs` of type ***String[]*** - List of bridge handles
- Input: `busInterfaceID` of type ***String*** - Handle to a busInterface element

F.7.24.17 getBusInterfaceSlaveFileSetGroupIDs

Description: Returns the fileSetGroupIDs for a given busInterface element (slave only)

- Returns: `fileSetGroupIDs` of type ***String[]*** - List of fileSetGroup handles
- Input: `busInterfaceID` of type ***String*** - Handle to a busInterface element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.24.18 getFileSetGroupName

Description: Returns the name for a given fileSetGroup element

- Returns: `name` of type ***String*** - FileSetGroup name
- Input: `fileSetGroupID` of type ***String*** - Handle to a fileSetGroup element

F.7.24.19 getFileSetGroupFileSetIDs

Description: Returns the fileSetIDs for a given fileSetGroup element

- Returns: `fileSetIDs` of type ***String[]*** - List of fileSet handles
- Input: `fileSetGroupID` of type ***String*** - Handle to a fileSetGroup element

F.7.24.20 getBusInterfaceGroupName

Description: Returns the groupName for a given busInterface element (system, mirroredSystem, and group only)

- Returns: `name` of type ***String*** - Group name
- Input: `busInterfaceID` of type ***String*** - Handle to a busInterface element

F.7.24.21 getBusInterfaceMirroredSlaveRemapAddressIDs

Description: Returns the remapAddressIDs for a given busInterface element (mirroredSlave only)

- Returns: `remapAddressIDs` of type ***String[]*** - List of remapAddress handles
- Input: `busInterfaceID` of type ***String*** - Handle to a busInterface element

F.7.24.22 getBusInterfaceMirroredSlaveRange

Description: Returns the ranges for a given busInterface element (mirroredSlave only)

- Returns: `range` of type ***Number*** - Range with left at index 0 and right at index 1
- Input: `busInterfaceID` of type ***String*** - Handle to a busInterface element

F.7.24.23 getBusInterfaceMonitorInterfaceMode

Description: Returns the monitor interfaceMode attribute value for a given busInterface element (monitor only)

- Returns: `value` of type ***String*** - InterfaceMode value
- Input: `busInterfaceID` of type ***String*** - Handle to a busInterface element

F.7.24.24 getBusInterfaceConnectionRequired

Description: Returns the connectionRequired for a given busInterface element

- Returns: `value` of type ***Boolean*** - ConnectionRequired value
- Input: `busInterfaceID` of type ***String*** - Handle to a busInterface element

F.7.24.25 getBusInterfaceBitSteering

Description: Returns the bitSteering for a given busInterface element

- Returns: `value` of type ***String*** - BitSteering value
- Input: `busInterfaceID` of type ***String*** - Handle to a busInterface element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.24.26 getBusInterfaceEndianness

Description: Returns the endianness for a given busInterface element

- Returns: `value` of type ***String*** - Endianness value
- Input: `busInterfaceID` of type ***String*** - Handle to a busInterface element

F.7.25 Businterface operations (EXTENDED)

F.7.25.1 addBusInterfaceAbstractionType

Description: Add an abstractionType with the given abstractionRef to the given busInterface element

- Returns: `abstractionTypeID` of type ***String*** - Handle to a new abstractionType element
- Input: `busInterfaceID` of type ***String*** - Handle to a busInterface element
- Input: `abstractionRef` of type ***String[]*** - AbstractionDef VLNV

F.7.25.2 removeBusInterfaceAbstractionType

Description: Remove the given abstractionType element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `abstractionTypeID` of type ***String*** - Handle to an abstractionType element

F.7.25.3 addAbstractionTypeViewRef

Description: Add a viewRef with the given name to the given abstractionType element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `abstractionTypeID` of type ***String*** - Handle to an abstractionType element
- Input: `viewRef` of type ***String*** - View name

F.7.25.4 removeAbstractionTypeViewRef

Description: Remove the viewRef element with the given viewName

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `abstractionTypeID` of type ***String*** - Handle to an abstractionType element
- Input: `viewRef` of type ***String*** - View name

F.7.25.5 addAbstractionTypePortMap

Description: Add a portMap with the given name, logicalPortName, and physicalPortNameOrTieValue to the given abstractionType

- Returns: `portMapID` of type ***String*** - Handle to a new portMap element
- Input: `abstractionTypeID` of type ***String*** - Handle to an abstractionType element
- Input: `logicalPortName` of type ***String*** - Logical port name
- Input: `physicalPortNameOrTieValue` of type ***String*** - Physical port name or logical tie off value

F.7.25.6 removeAbstractionTypePortMap

Description: Remove the given portMap element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

- Input: `portMapID` of type ***String*** - Handle to a portMap element

F.7.25.7 setPortMapRange

Description: Set the range of the given logicalPort or physicalPort element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `logicalPortID` | `physicalPortID` of type ***String*** - logicalPortID | physicalPortID
- Input: `range` of type ***String[]*** - Range with left at index 0 and right at index 1

F.7.25.8 setPortMapIndices

Description: Set the indices for the given physicalPort element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `physicalPortID` of type ***String*** - Handle to a physicalPort element
- Input: `indices` of type ***String[]*** - List of indices

F.7.25.9 setPortMapIsInformative

Description: Set the isInformative for the given portMap element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `portMapID` of type ***String*** - Handle to a portMap element
- Input: `value` of type ***Boolean*** - IsInformative value

F.7.25.10 setBusInterfaceMasterAddressSpaceName

Description: Set the addressSpaceRef with the given name for the given busInterface element (master only)

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `busInterfaceID` of type ***String*** - Handle to a busInterface element
- Input: `name` of type ***String*** - AddressSpace name

F.7.25.11 setBusInterfaceMasterBaseAddress

Description: Set the baseAddress with the given expression for the given busInterface element (master only)

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `busInterfaceID` of type ***String*** - Handle to a busInterface element
- Input: `baseAddress` of type ***String*** - baseAddress expression

F.7.25.12 setBusInterfaceSlaveMemoryMapName

Description: Set the memoryMapRef with the given name for the given busInterface element (slave only)

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `busInterfaceID` of type ***String*** - Handle to a busInterface element
- Input: `name` of type ***String*** - MemoryMap name

F.7.25.13 addBusInterfaceSlaveBridge

Description: Add a transparentBridge with the given name for the given busInterface (slave only)

- Returns: `bridgeID` of type ***String*** - Handle to a new bridge element
- Input: `busInterfaceID` of type ***String*** - Handle to a busInterface element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

— Input: `name` of type ***String*** - BusInterface name for masterRef attribute

F.7.25.14 removeBusInterfaceSlaveBridge

Description: Remove the given bridge element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

— Input: `bridgeID` of type ***String*** - Handle to a bridge element

F.7.25.15 addBusInterfaceSlaveFileSetGroup

Description: Add a fileSetGroup with the given name and fileSetRefs to the given busInterface element (slave only)

— Returns: `fileSetGroupID` of type ***String*** - Handle to a new fileSetGroup element

— Input: `busInterfaceID` of type ***String*** - Handle to a busInterface element

— Input: `group` of type ***String*** - Group name

— Input: `fileSetRefs` of type ***String[]*** - List of fileSet names

F.7.25.16 removeBusInterfaceSlaveFileSetGroup

Description: Remove the given fileSetGroup

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

— Input: `fileSetGroupID` of type ***String*** - Handle to a fileSetGroup element

F.7.25.17 setBusInterfaceGroupName

Description: Set the group with the given name for the given busInterface element (system, mirroredSystem, and group only)

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

— Input: `busInterfaceID` of type ***String*** - Handle to a busInterface element

— Input: `group` of type ***String*** - group name

F.7.25.18 setBusInterfaceConnectionRequired

Description: Set connectionRequired with the given value for the given busInterface element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

— Input: `busInterfaceID` of type ***String*** - Handle to a busInterface element

— Input: `value` of type ***Boolean*** - ConnectionRequired value

F.7.25.19 setBusInterfaceBitSteering

Description: Set bitSteering with the given value for the given busInterface element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

— Input: `busInterfaceID` of type ***String*** - Handle to a busInterface element

— Input: `value` of type ***String*** - BitSteering value

F.7.25.20 setBusInterfaceEndianness

Description: Set endianness with the given value for the given busInterface element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

- Input: `busInterfaceID` of type ***String*** - Handle to a busInterface element
- Input: `value` of type ***String*** - Endianness value

F.7.26 Catalog operations (EXTENDED)

F.7.26.1 getCatalogCatalogsIpxactFiles

Description: Returns catalogs IpxactFileIDs for the given catalog element

- Returns: `ipxactFileIDs` of type ***String[]*** - List of ipxactFile handles
- Input: `catalogID` of type ***String*** - Handle to a catalog element

F.7.26.2 getIpxactFileVLNV

Description: Returns VLNV for the given ipxactFile element

- Returns: `VLNV` of type ***String[]*** - VLNV
- Input: `ipxactFileID` of type ***String*** - Handle to an ipxactFile element

F.7.26.3 getIpxactFileName

Description: Returns name for the given ipxactFile element

- Returns: `fileName` of type ***String*** - File name
- Input: `ipxactFileID` of type ***String*** - Handle to an ipxactFile element

F.7.26.4 getCatalogBusDefinitionsIpxactFiles

Description: Returns busDefintions IpxactFileIDs for the given catalog element

- Returns: `ipxactFileIDs` of type ***String[]*** - List of ipxactFile handles
- Input: `catalogID` of type ***String*** - Handle to a catalog element

F.7.26.5 getCatalogAbstractionDefinitionsIpxactFiles

Description: Returns abstractionDefintions IpxactFileIDs for the given catalog element

- Returns: `ipxactFileIDs` of type ***String[]*** - List of ipxactFile handles
- Input: `catalogID` of type ***String*** - Handle to a catalog element

F.7.26.6 getCatalogComponentsIpxactFiles

Description: Returns components IpxactFileIDs for the given catalog element

- Returns: `ipxactFileIDs` of type ***String[]*** - List of ipxactFile handles
- Input: `catalogID` of type ***String*** - Handle to a catalog element

F.7.26.7 getCatalogAbstractorsIpxactFiles

Description: Returns abstractors IpxactFileIDs for the given catalog element

- Returns: `ipxactFileIDs` of type ***String[]*** - List of ipxactFile handles
- Input: `catalogID` of type ***String*** - Handle to a catalog element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.26.8 getCatalogDesignsIpxactFiles

Description: Returns designs IpxactFileIDs for the given catalog element

- Returns: `ipxactFileIDs` of type ***String[]*** - List of ipxactFile handles
- Input: `catalogID` of type ***String*** - Handle to a catalog element

F.7.26.9 getCatalogDesignConfigurationsIpxactFiles

Description: Returns designConfigurations IpxactFileIDs for the given catalog element

- Returns: `ipxactFileIDs` of type ***String[]*** - List of ipxactFile handles
- Input: `catalogID` of type ***String*** - Handle to a catalog element

F.7.26.10 getCatalogGeneratorChainsIpxactFiles

Description: Returns generatorChains IpxactFileIDs for the given catalog element

- Returns: `ipxactFileIDs` of type ***String[]*** - List of ipxactFile handles
- Input: `catalogID` of type ***String*** - Handle to a catalog element

F.7.26.11 createCatalog

Description: Create a new catalog and returns its catalogID; fails and returns null if VLNV already exists

- Returns: `catalogID` of type ***String*** - Handle to a new catalog element
- Input: `catalogVLNV` of type ***String[]*** - Catalog VLNV

F.7.26.12 addCatalogCatalogsIpxactFiles

Description: Add ipxactFile with the given VLNV and name to catalogs in the given catalog element

- Returns: `ipxactFileID` of type ***String*** - Handle to an ipxactFile element
- Input: `catalogID` of type ***String*** - Handle to a catalog element
- Input: `catalogVLNV` of type ***String[]*** - Catalog VLNV
- Input: `fileName` of type ***String*** - Catalog file name

F.7.26.13 removeCatalogCatalogsIpxactFiles

Description: Remove the given ipxactFile element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `ipxactFileID` of type ***String*** - Handle to an ipxactFile element

F.7.26.14 addCatalogBusDefinitionsIpxactFiles

Description: Add ipxactFile with the given VLNV and name to busDefinitions in the given catalog element

- Returns: `ipxactFileID` of type String - Handle to an ipxactFile element
- Input: `catalogID` of type ***String*** - Handle to a catalog element
- Input: `busDefVLNV` of type ***String[]*** - BusDef VLNV
- Input: `fileName` of type ***String*** - Bus definition file name

F.7.26.15 removeCatalogBusDefinitionsIpxactFiles

Description: Remove the given ipxactFile element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `ipxactFileID` of type ***String*** - Handle to an ipxactFile element

F.7.26.16 addCatalogAbstractionDefinitionsIpxactFiles

Description: Add ipxactFile with the given VLNV and name to abstractionDefinitions in the given catalog element

- Returns: `ipxactFileID` of type ***String*** - Handle to an ipxactFile element
- Input: `catalogID` of type ***String*** - Handle to a catalog element
- Input: `abstractionDefVLNV` of type ***String[]*** - AbstractionDef VLNV
- Input: `fileName` of type ***String*** - Abstraction definition file name

F.7.26.17 removeCatalogAbstractionDefinitionsIpxactFiles

Description: Remove the given ipxactFile element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `ipxactFileID` of type ***String*** - Handle to an ipxactFile element

F.7.26.18 addCatalogComponentsIpxactFiles

Description: Add ipxactFile with the given VLNV and name to components in the given catalog element

- Returns: `ipxactFileID` of type ***String*** - Handle to an ipxactFile element
- Input: `catalogID` of type ***String*** - Handle to a catalog element
- Input: `componentVLNV` of type ***String[]*** - Component VLNV
- Input: `fileName` of type ***String*** - Component file name

F.7.26.19 removeCatalogComponentsIpxactFiles

Description: Remove the given ipxactFile element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `ipxactFileID` of type ***String*** - Handle to an ipxactFile element

F.7.26.20 addCatalogAbstractorsIpxactFiles

Description: Add ipxactFile with the given VLNV and name to abstractors in the given catalog element

- Returns: `ipxactFileID` of type ***String*** - Handle to an ipxactFile element
- Input: `catalogID` of type ***String*** - Handle to a catalog element
- Input: `abstractorVLNV` of type ***String[]*** - Abstractor VLNV
- Input: `fileName` of type ***String*** - Abstractor file name

F.7.26.21 removeCatalogAbstractorsIpxactFiles

Description: Remove the given ipxactFile element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `ipxactFileID` of type ***String*** - Handle to an ipxactFile element

F.7.26.22 addCatalogDesignsIpxactFiles

Description: Add ipxactFile with the given VLNV and name to designs in the given catalog element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

- Returns: `ipxactFileID` of type ***String*** - Handle to an ipxactFile element
- Input: `catalogID` of type ***String*** - Handle to a catalog element
- Input: `designVLNV` of type ***String[]*** - Design VLNV
- Input: `fileName` of type ***String*** - Design file name

F.7.26.23 removeCatalogDesignsIpxactFiles

Description: Remove the given ipxactFile element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `ipxactFileID` of type ***String*** - Handle to an ipxactFile element

F.7.26.24 addCatalogDesignConfigurationsIpxactFiles

Description: Add ipxactFile with the given VLNV and name to designConfigurations in the given catalog element

- Returns: `ipxactFileID` of type ***String*** - Handle to an ipxactFile element
- Input: `catalogID` of type ***String*** - Handle to a catalog element
- Input: `designConfigurationVLNV` of type ***String[]*** - DesignConfiguration VLNV
- Input: `fileName` of type ***String*** - Design configuration file name

F.7.26.25 removeCatalogDesignConfigurationsIpxactFiles

Description: Remove the given ipxactFile element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `ipxactFileID` of type ***String*** - Handle to an ipxactFile element

F.7.26.26 addCatalogGeneratorChainsIpxactFiles

Description: Add ipxactFile with the given VLNV and name to generatorChains in the given catalog element

- Returns: `ipxactFileID` of type ***String*** - Handle to an ipxactFile element
- Input: `catalogID` of type ***String*** - Handle to a catalog element
- Input: `generatorChainVLNV` of type ***String[]*** - GeneratorChain VLNV
- Input: `fileName` of type ***String*** - Generator chain file name

F.7.26.27 removeCatalogGeneratorChainsIpxactFiles

Description: Remove the given ipxactFile element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `ipxactFileID` of type ***String*** - Handle to an ipxactFile element

F.7.27 Channel operations (BASE)

F.7.27.1 getChannelBusInterfaceIDs

Description: Returns the busInterfaceIDs for the given channel element

- Returns: `busInterfaceIDs` of type ***String[]*** - List of busInterfaceIDs
- Input: `channelID` of type ***String*** - Handle to a channel element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.28 Channel operations (EXTENDED)

F.7.28.1 addChannelBusInterface

Description: Add a busInterfaceRef with the given name to the given channel element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `channelID` of type ***String*** - Handle to a channel element
- Input: `name` of type ***String*** - BusInterface name

F.7.28.2 removeChannelBusInterface

Description: Remove the busInterfaceRef with the given name from the given channel element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `channelID` of type ***String*** - Handle to a channel element
- Input: `name` of type ***String*** - BusInterface name

F.7.29 Choice operations (BASE)

F.7.29.1 getChoiceEnumerationIDs

Description: Returns choiceEnumerationIDs for the given choice element

- Returns: `choiceEnumerationIDs` of type ***String[]*** - List of choiceEnumeration handles
- Input: `choiceID` of type ***String*** - Handle to a choice element

F.7.29.2 getEnumerationText

Description: Returns text for the given choice element

- Returns: `text` of type ***String*** - Enumeration tex
- Input: `choiceEnumerationID` of type ***String*** - Handle to a choiceEnumeration element

F.7.29.3 getEnumerationHelp

Description: Returns help for the given choice element

- Returns: `help` of type ***String*** - Enumeration help
- Input: `choiceEnumerationID` of type ***String*** - Handle to a choiceEnumeration element

F.7.30 Choice operations (EXTENDED)

F.7.30.1 addChoiceEnumeration

Description: Add enumeration with the given name to the given choice element

- Returns: `choiceEnumerationID` of type ***String*** - Handle to a new choiceEnumeration
- Input: `choiceID` of type ***String*** - Handle to a choice element
- Input: `name` of type ***String*** - ChoiceEnumeration name

F.7.30.2 removeChoiceEnumeration

Description: Remove the given choiceEnumeration

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `choiceEnumerationID` of type ***String*** - Handle to a choiceEnumeration element

F.7.30.3 setEnumerationText

Description: Set text with the given value for the given choiceEnumeration element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `choiceEnumerationID` of type ***String*** - Handle to a choiceEnumeration element
— Input: `text` of type ***String*** - ChoiceEnumeration text

F.7.30.4 setEnumerationHelp

Description: Set help with the given value for the given choiceEnumeration element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `choiceEnumerationID` of type ***String*** - Handle to a choiceEnumeration element
— Input: `help` of type ***String*** - ChoiceEnumeration help

F.7.31 Component instantiation operations (BASE)

F.7.31.1 getComponentInstantiationIsVirtual

Description: Returns isVirtual for the given componentInstantiation element

— Returns: `value` of type ***Boolean*** - ComponentInstantiation isVirtual value
— Input: `componentInstantiationID` of type ***String*** - Handle to a componentInstantiation element

F.7.31.2 getComponentInstantiationLanguage

Description: Returns language for the given componentInstantiation element

— Returns: `language` of type ***String*** - ComponentInstantiation language
— Input: `componentInstantiationID` of type ***String*** - Handle to a componentInstantiation element

F.7.31.3 getComponentInstantiationLibraryName

Description: Returns libraryName for the given componentInstantiation element

— Returns: `libraryName` of type ***String*** - ComponentInstantiation libraryName
— Input: `componentInstantiationID` of type ***String*** - Handle to a componentInstantiation element

F.7.31.4 getComponentInstantiationPackageName

Description: Returns packageName for the given componentInstantiation element

— Returns: `packageName` of type ***String*** - ComponentInstantiation packageName
— Input: `componentInstantiationID` of type ***String*** - Handle to a componentInstantiation element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.31.5 getComponentInstantiationModuleName

Description: Returns moduleName for the given componentInstantiation element

- Returns: `moduleName` of type ***String*** - ComponentInstantiation moduleName
- Input: `componentInstantiationID` of type ***String*** - Handle to a componentInstantiation element

F.7.31.6 getComponentInstantiationArchitectureName

Description: Returns architectureName for the given componentInstantiation element

- Returns: `architectureName` of type ***String*** - ComponentInstantiation architectureName
- Input: `componentInstantiationID` of type ***String*** - Handle to a componentInstantiation element

F.7.31.7 getComponentInstantiationConfigurationName

Description: Returns configurationName for the given componentInstantiation element

- Returns: `configurationName` of type ***String*** - ComponentInstantiation configurationName
- Input: `componentInstantiationID` of type ***String*** - Handle to a componentInstantiation element

F.7.31.8 getComponentInstantiationDefaultFileBuilderIDs

Description: Returns defaultFileBuilderIDs for the given componentInstantiation element

- Returns: `fileBuilderIDs` of type ***String[]*** - List of fileBuilder handles
- Input: componentInstantiationID of type ***String*** - Handle to a componentInstantiation element

F.7.31.9 getComponentInstantiationFileSetRefIDs

Description: Returns fileSetRefIDs for the given componentInstantiation element

- Returns: `fileSetRefIDs` of type ***String[]*** - List of fileSetRef handles
- Input: `componentInstantiationID` of type ***String*** - Handle to a componentInstantiation element

F.7.31.10 getFileSetRefLocalName

Description: Returns locaName for the given fileSetRef element

- Returns: `localName` of type ***String*** - FileSetRef localName
- Input: `fileSetRefID` of type ***String*** - Handle to a fileSetRef element

F.7.31.11 getComponentInstantiationContraintSetRefIDs

Description: Returns constraintSetRefIDs for the given componentInstantiation element

- Returns: `contraintSetRefIDs` of type ***String[]*** - List of constraintSetRef handles
- Input: `componentInstantiationID` of type ***String*** - Handle to a componentInstantiation element

F.7.31.12 getConstraintSetRefLocalName

Description: Returns locaName for the given constraintSetRefRef element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

— Returns: `localName` of type ***String*** - ConstraintSetRef localName
— Input: `constraintSetRefID` of type ***String*** - Handle to a constraintSetRef element

F.7.31.13 getComponentInstantiationWhiteboxElementRefIDs

Description: Returns whiteboxElementRefIDs for the given componentInstantiation element

— Returns: `whiteboxElementRefIDs` of type ***String[]*** - List of whiteboxElementRef handles
— Input: `componentInstantiationID` of type ***String*** - Handle to a componentInstantiation element

F.7.32 Component instantiation operations (EXTENDED)

F.7.32.1 setComponentInstantiationIsVirtual

Description: Set isVirtual with the given value for the given componentInstantiation element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `componentInstantiationID` of type ***String*** - Handle to a componentInstantiation element
— Input: `value` of type ***Boolean*** - isVirtual value

F.7.32.2 setComponentInstantiationLanguage

Description: Set language with the given value for the given componentInstantiation element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `componentInstantiationID` of type ***String*** - Handle to a componentInstantiation element
— Input: `language` of type ***String*** - Language value

F.7.32.3 setComponentInstantiationLibraryName

Description: Set libraryName with the given value for the given componentInstantiation element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `componentInstantiationID` of type ***String*** - Handle to a componentInstantiation element
— Input: `libraryName` of type ***String*** - LibraryName value

F.7.32.4 setComponentInstantiationPackageName

Description: Set packageName with the given value for the given componentInstantiation element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `componentInstantiationID` of type ***String*** - Handle to a componentInstantiation element
— Input: `packageName` of type ***String*** - PackageName value

F.7.32.5 setComponentInstantiationModuleName

Description: Set moduleName with the given value for the given componentInstantiation element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

- Input: `componentInstantiationID` of type ***String*** - Handle to a componentInstantiation element
- Input: `moduleName` of type ***String*** - ModuleName value

F.7.32.6 setComponentInstantiationArchitectureName

Description: Set architectureName with the given value for the given componentInstantiation element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `componentInstantiationID` of type ***String*** - Handle to a componentInstantiation element
- Input: `architectureName` of type ***String*** - ArchitectureName value

F.7.32.7 setComponentInstantiationConfigurationName

Description: Set configurationName with the given value for the given componentInstantiation element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `componentInstantiationID` of type ***String*** - Handle to a componentInstantiation element
- Input: `configurationName` of type ***String*** - ConfigurationName value

F.7.32.8 addComponentInstantiationDefaultFileBuilder

Description: Add defaultFileBuilder with the given fileType to the given componentInstantiation element

- Returns: `fileBuilderID` of type ***String*** - Handle to a new fileBuilder
- Input: `componentInstantiationID` of type ***String*** - Handle to a componentInstantiation element
- Input: `fileType` of type ***String*** - DefaultFileBuilder fileType

F.7.32.9 removeComponentInstantiationDefaultFileBuilder

Description: Remove the given defaultFileBuilder element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `fileBuilderID` of type ***String*** - Handle to a fileBuilder element

F.7.32.10 addComponentInstantiationFileSetRef

Description: Add fileSetRef with the given localName to the given componentInstantiation element

- Returns: `fileSetRefID` of type ***String*** - Handle to a new fileSetRef
- Input: `componentInstantiationID` of type ***String*** - Handle to a componentInstantiation element
- Input: `localName` of type ***String*** - FileSetRef localName

F.7.32.11 removeComponentInstantiationFileSetRef

Description: Remove the given fileSetRef element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `fileSetRefID` of type ***String*** - Handle to a fileSetRef element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.32.12 addComponentInstantiationConstraintSetRef

Description: Add constraintRefSet with the given localName to the given componentInstantiation element

- Returns: `constraintSetRefID` of type ***String*** - Handle to a new constraintSetRef
- Input: `componentInstantiationID` of type ***String*** - Handle to a componentInstantiation element
- Input: `localName` of type ***String*** - ConstraintSetRef localName

F.7.32.13 removeComponentInstantiationConstraintSetRef

Description: Remove the given constraintSetRef element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `constraintSetRefID` of type ***String*** - Handle to a constraintSetRef element

F.7.32.14 addComponentInstantiationWhiteboxElementRef

Description: Add whiteboxElementRef with the given name, pathSegmentName, and indices to the given componentInstantiation element

- Returns: `whiteboxElementRefID` of type ***String*** - Handle to a new whiteboxElementRef
- Input: `componentInstantiationID` of type ***String*** - Handle to a componentInstantiation element
- Input: `name` of type ***String*** - WhiteboxElementRef name
- Input: `pathSegmentName` of type ***String*** - WhiteboxElementRef pathSegment name
- Input: `indices` of type ***String[]*** - WhiteboxElement indices

F.7.32.15 removeComponentInstantiationWhiteboxElementRef

Description: Remove the given whiteboxElementRef element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `whiteboxElementRefID` of type ***String*** - Handle to a whiteboxElementRef element

F.7.33 Component operations (BASE)

F.7.33.1 getComponentBusInterfaceIDs

Description: Returns the busInterfaceIDs for a given component or component instance element

- Returns: `busInterfaceIDs` of type ***String[]*** - List of busInterface handles
- Input: `componentID` | `componentInstanceID` of type ***String*** - Handle to a component or componentInstance element

F.7.33.2 getComponentIndirectInterfaceIDs

Description: Returns the indirectInterfaceIDs for a given component or component instance element

- Returns: `indirectInterfaceIDs` of type ***String[]*** - List of indirectInterface handles
- Input: `componentID` | `componentInstanceID` of type ***String*** - Handle to a component or componentInstance element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.33.3 getComponentChannelIDs

Description: Returns the channelIDs for a given component or component instance element

- Returns: `channelIDs` of type ***String[]*** - List of channel handles
- Input: `componentID` | componentInstanceID of type ***String*** - Handle to a component or componentInstance element

F.7.33.4 getComponentRemapStateIDs

Description: Returns the remapStateIDs for a given component or component instance element

- Returns: `remapStateIDs` of type ***String[]*** - List of remapState handles
- Input: `componentID` | `componentInstanceID` of type ***String*** - Handle to a component or componentInstance element

F.7.33.5 getComponentAddressSpaceIDs

Description: Returns the addressSpaceIDs for a given component or component instance element

- Returns: `addressSpaceIDs` of type ***String[]*** - List of addressSpace handles
- Input: `componentID` | `componentInstanceID` of type ***String*** - Handle to a component or componentInstance element

F.7.33.6 getComponentMemoryMapIDs

Description: Returns the memoryMapIDs for a given component or component instance element

- Returns: `memoryMapIDs` of type ***String[]*** - List of memoryMap handles
- Input: `componentID` | `componentInstanceID` of type ***String*** - Handle to a component or componentInstance element

F.7.33.7 getComponentViewIDs

Description: Returns the viewIDs for a given component or component instance element

- Returns: `viewIDs` of type ***String[]*** - List of view handles
- Input: `componentID` | `componentInstanceID` of type ***String*** - Handle to a component or componentInstance element

F.7.33.8 getComponentSelectedViewIDs

Description: Returns the selected viewIDs for a given component or component instance element

- Returns: `viewIDs` of type ***String[]*** - List of view handles (list contains selected views only)
- Input: `componentID` | `componentInstanceID` of type ***String*** - Handle to a component or componentInstance element

F.7.33.9 getComponentComponentInstantiationIDs

Description: Returns the componentInstantiationIDs for a given component or component instance element

- Returns: `componentInstantiationIDs` of type ***String[]*** - List of componentInstantiation handles
- Input: `componentID` | `componentInstanceID` of type ***String*** - Handle to a component or componentInstance element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.33.10 getComponentDesignInstantiationIDs

Description: Returns the designInstantiationIDs for a given component or component instance element

- Returns: `designInstantiationIDs` of type ***String[]*** - List of designInstantiation handles
- Input: `componentID` | `componentInstanceID` of type ***String*** - Handle to a component or componentInstance element

F.7.33.11 getComponentDesignConfigurationInstantiationIDs

Description: Returns the designConfigurationInstantiationIDs for a given component or component instance element

- Returns: `designConfigurationInstantiationIDs` of type ***String[]*** - List of designConfigurationInstantiation handles
- Input: `componentID` | `componentInstanceID` of type ***String*** - Handle to a component or componentInstance element

F.7.33.12 getComponentPortIDs

Description: Returns the portIDs for a given component or component instance element

- Returns: `portIDs` of type ***String[]*** - List of portID handles
- Input: `componentID` | `componentInstanceID` of type ***String*** - Handle to a component or componentInstance element

F.7.33.13 getComponentGeneratorIDs

Description: Returns the generatorIDs for a given component or component instance element

- Returns: `generatorIDs` of type ***String[]*** - List of generatorID handles
- Input: `componentID` | `componentInstanceID` of type ***String*** - Handle to a component or componentInstance element

F.7.33.14 getComponentChoiceIDs

Description: Returns the choiceIDs for a given component or component instance element

- Returns: `choiceIDs` of type ***String[]*** - List of choiceID handles
- Input: `componentID` | `componentInstanceID` of type ***String*** - Handle to a component or componentInstance element

F.7.33.15 getComponentFileSetIDs

Description: Returns the fileSetIDs for a given component or component instance element

- Returns: `fileSetIDs` of type ***String[]*** - List of fileSetID handles
- Input: `componentID` | `componentInstanceID` of type ***String*** - Handle to a component or componentInstance element

F.7.33.16 getComponentWhiteboxElementIDs

Description: Returns the whiteboxElementIDs for a given component or component instance element

- Returns: `whiteboxElementIDs` of type ***String[]*** - List of whiteboxElement handles
- Input: `componentID` | `componentInstanceID` of type ***String*** - Handle to a component or componentInstance element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.33.17 getComponentCpuIDs

Description: Returns the cpuIDs for a given component or component instance element

- Returns: `cpuIDs` of type ***String[]*** - List of cpu handles
- Input: `componentID` | `componentInstanceID` of type ***String*** - Handle to a component or componentInstance element

F.7.33.18 getComponentOtherClockDriverIDs

Description: Returns the clockDriverIDs for a given component or component instance element

- Returns: `clockDriverIDs` of type ***String[]*** - List of clockDriver handles
- Input: `componentID` | `componentInstanceID` of type ***String*** - Handle to a component or componentInstance element

F.7.33.19 getComponentResetTypeIDs

Description: Returns the resetTypeIDs for a given component or component instance element

- Returns: `resetTypeIDs` of type ***String[]*** - List of resetType handles
- Input: `componentID` | `componentInstanceID` of type ***String*** - Handle to a component or componentInstance element

F.7.34 Component operations (EXTENDED)

F.7.34.1 createComponent

Description: Create a new component with the given VLNV and returns its componentID; fails and returns null if VLNV already exists

- Returns: `componentID` of type ***String*** - Handle to a new component
- Input: `componentVLNV` of type ***String[]*** - VLNV for a new component

F.7.34.2 addComponentBusInterface

Description: Add a busInterface with the given name, busType, interfaceMode, and group to the given component element

- Returns: `busInterfaceID` of type ***String*** - Handle to a new busInterface
- Input: `componentID` of type ***String*** - Handle to a component element
- Input: `name` of type ***String*** - BusInterface name
- Input: `busVLNV` of type ***String[]*** - BusDef VLNV
- Input: `interfaceMode` of type ***String*** - InterfaceMode string
- Input: `group` of type ***String*** - group

F.7.34.3 removeComponentBusInterface

Description: Remove the given busInterface element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `busInterfaceID` of type ***String*** - Handle to a busInterface element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.34.4 addComponentIndirectInterface

Description: Add an indirectInterface with the given name, indirectAddress, indirectData, memoryMapRef or busInterfaceRef to the given component element

- Returns: `indirectInterfaceID` of type ***String*** - Handle to a new indirectInterface
- Input: `componentID` of type ***String*** - Handle to a component element
- Input: `name` of type ***String*** - IndirectInterface name
- Input: `indirectAddress` of type ***String[]*** - IndirectAddress value
- Input: `indirectData` of type ***String[]*** - IndirectData value
- Input: `memoryMapRef` of type ***String*** - Name of memoryMap or null
- Input: `busInterfaceRef` of type ***String*** - Name of busInterface or null

F.7.34.5 removeComponentIndirectInterface

Description: Remove the given indirectInterface element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `indirectInterfaceID` of type ***String*** - Handle to an indirectInterface element

F.7.34.6 addComponentChannel

Description: Add a channel with the given name and busInterfaceRefs to the given component element

- Returns: `channelID` of type ***String*** - Handle to a new channel
- Input: `componentID` of type ***String*** - Handle to a component element
- Input: `name` of type ***String*** - Channel name
- Input: `busInterfaceRefs` of type ***String[]*** - List of busInterface names

F.7.34.7 removeComponentChannel

Description: Remove the given channel element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `channelID` of type ***String*** - Handle to a channel element

F.7.34.8 addComponentRemapState

Description: Add a remapState with the given name to the given component

- Returns: `remapStateID` of type ***String*** - Handle to a new remapState
- Input: `componentID` of type ***String*** - Handle to a component element
- Input: `name` of type ***String*** - RemapState name

F.7.34.9 removeComponentRemapState

Description: Remove the given remapState element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `remapStateID` of type ***String*** - Handle to a remapState element

F.7.34.10 addComponentAddressSpace

Description: Add an addressSpace with the given name, range, and width to the given component element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

- Returns: `addressSpaceID` of type ***String*** - Handle to a new addressSpace
- Input: `componentID` of type ***String*** - Handle to a component element
- Input: `name` of type ***String*** - AddressSpace name
- Input: `range` of type ***Number*** - AddressSpace range
- Input: `width` of type ***Number*** - AddrressSpace width

F.7.34.11 removeComponentAddressSpace

Description: Remove the given addressSpace element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `addressSpaceID` of type ***String*** - Handle to an addressSpace element

F.7.34.12 addComponentMemoryMap

Description: Add a memoryMap with the given name to the given component element

- Returns: `memoryMapID` of type ***String*** - Handle to a new memoryMap
- Input: `componentID` of type ***String*** - Handle to a component element
- Input: `name` of type ***String*** - MemoryMap name

F.7.34.13 removeComponentMemoryMap

Description: Remove the given memoryMap element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `memoryMapID` of type ***String*** - Handle to a memoryMap element

F.7.34.14 addComponentView

Description: Add a view with the given name and envIdentifier to the given component element

- Returns: `viewID` of type ***String*** - Handle to a new view
- Input: `componentID` of type ***String*** - Handle to a component element
- Input: `name` of type ***String*** - View name
- Input: `envIdentifier` of type ***String[]*** - EnvIdentifier value

F.7.34.15 removeComponentView

Description: Remove the given view element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `viewID` of type ***String*** - Handle to a view element

F.7.34.16 addComponentComponentInstantiation

Description: Add a componentInstantiation with the given name to the given component element

- Returns: `componentInstantiationID` of type ***String*** - Handle to a new componentInstantiation
- Input: `componentID` of type ***String*** - Handle to a component element
- Input: `name` of type ***String*** - ComponentInstantiation name

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.34.17 removeComponentComponentInstantiation

Description: Remove the given componentInstantiation element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `componentInstantiationID` of type ***String*** - Handle to a componentInstantiation element

F.7.34.18 addComponentDesignInstantiation

Description: Add a designInstantiation with the given name and designRef to the given component element

- Returns: `designInstantiationID` of type ***String*** - Handle to a new designInstantiation
- Input: `componentID` of type ***String*** - Handle to a component element
- Input: `name` of type ***String*** - DesignInstantiation name
- Input: `designVLNV` of type ***String[]*** - Design VLNV

F.7.34.19 removeComponentDesignInstantiation

Description: Remove the given designInstantiation element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `designInstantiationID` of type ***String*** - Handle to a designInstantiation element

F.7.34.20 addComponentDesignConfigurationInstantiation

Description: Add a designConfigurationInstantiation with the given name and designConfigurationRef to the given component element

- Returns: `designConfigurationInstantiationID` of type ***String*** - Handle to a new designConfigurationInstantiation
- Input: `componentID` of type ***String*** - Handle to a component element
- Input: `name` of type ***String*** - DesignConfigurationInstantiation name
- Input: `designConfigurationVLNV` of type ***String[]*** - DesignConfiguration VLNV

F.7.34.21 removeComponentDesignConfigurationInstantiation

Description: Remove the given designConfigurationInstantiation element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `designConfigurationInstantiationID` of type ***String*** - Handle to a designConfigurationInstantiation element

F.7.34.22 addComponentPort

Description: Add a port with the given name, type, and directionOrInitiative to the given component element

- Returns: `portID` of type ***String*** - Handle to a new port
- Input: `componentID` of type ***String*** - Handle to a component element
- Input: `name` of type ***String*** - Port name
- Input: `type` of type ***String*** - Port wire type (wire or transactional)
- Input: `directionOrInitiative` of type ***String*** - Port direction or initiative

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.34.23 removeComponentPort

Description: Remove the given port element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `portID` of type ***String*** - Handle to a port element

F.7.34.24 addComponentGenerator

Description: Add a generator with the given name and path to the given component element

- Returns: `generatorID` of type ***String*** - Handle to a new generator
- Input: `componentID` of type ***String*** - Handle to a component element
- Input: `name` of type ***String*** - Generator name
- Input: `generatorExecutable` of type ***String*** - Path to generator executable

F.7.34.25 removeComponentGenerator

Description: Remove the given generator element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `generatorID` of type ***String*** - Handle to a generator element

F.7.34.26 addComponentChoice

Description: Add a choice with the given name and enumerations to the given component element

- Returns: `choiceID` of type ***String*** - Handle to a new choice
- Input: `componentID` of type ***String*** - Handle to a component element
- Input: `name` of type ***String*** - Choice name
- Input: `enumerations` of type ***String[]*** - List of enumeration values

F.7.34.27 removeComponentChoice

Description: Remove the given choice element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `choiceID` of type ***String*** - Handle to a choice element

F.7.34.28 addComponentFileSet

Description: Add a fileSet with the given name to the given component element

- Returns: `fileSetID` of type ***String*** - Handle to a new fileSet
- Input: `componentID` of type ***String*** - Handle to a component element
- Input: `name` of type ***String*** - FileSet name

F.7.34.29 removeComponentFileSet

Description: Remove the given fileSet element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `fileSetID` of type ***String*** - Handle to a fileSet element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.34.30 addComponentWhiteboxElement

Description: Add a whiteboxElement with the given name and type to the given component element

- Returns: `whiteboxElementID` of type ***String*** - Handle to a new whiteboxElement
- Input: `componentID` of type ***String*** - Handle to a component element
- Input: `name` of type ***String*** - WhiteboxElement name
- Input: `type` of type ***String*** - Whitebox element type

F.7.34.31 removeComponentWhiteboxElement

Description: Remove the given whiteboxElement element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `whiteboxElementID` of type ***String*** - Handle to a whiteboxElement element

F.7.34.32 addComponentCpu

Description: Add a cpu with the given name to the given component element

- Returns: `cpuID` of type ***String*** - Handle to a new cpu
- Input: `componentID` of type ***String*** - Handle to a component element
- Input: `name` of type ***String*** - Cpu name
- Input: `addressSpaceRefs` of type ***String[]*** - List of addressSpace names

F.7.34.33 removeComponentCpu

Description: Remove the given cpu element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `cpuID` of type ***String*** - Handle to a cpu

F.7.34.34 addComponentOtherClockDriver

Description: Add a clockDriver with the given name, period, offset, value, and duration to the given component element

- Returns: `clockDriverID` of type ***String*** - Handle to a new clockDriver
- Input: `componentID` of type ***String*** - Handle to a component element
- Input: `name` of type ***String*** - ClockDriver name
- Input: `period` of type ***Number*** - Clock period
- Input: `offset` of type ***Number*** - Clock pulse offset
- Input: `value` of type ***Number*** - Clock pulse value
- Input: `duration` of type ***Number*** - Clock pulse duration

F.7.34.35 removeComponentOtherClockDriver

Description: Remove the given clockDriver element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `clockDriverID` of type ***String*** - Handle to a clockDriver element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.34.36 addComponentResetType

Description: Add a resetType with the given name to the given component element

- Returns: `resetTypeID` of type ***String*** - Handle to a new resetType
- Input: `componentID` of type ***String*** - Handle to a component element
- Input: `name` of type ***String*** - ResetType name

F.7.34.37 removeComponentResetType

Description: Remove the given resetType element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `resetTypeID` of type ***String*** - Handle to a resetType element

F.7.35 Configurable element operations (BASE)

F.7.35.1 getConfigurableElementIDs

Description: Returns all configurableElementIDs of the given unconfigured element

- Returns: `configurableElementIDs` of type ***String[]*** - List of configurable element handles
- Input: `unconfiguredElementID` of type ***String*** - Handle to an unconfigured element

F.7.35.2 getConfigurableElementValue

Description: Returns the value of a given configurable element (default value)

- Returns: `value` of type ***String*** - Default configurable element value
- Input: `configurableElementID` of type ***String*** - Handle to a configurable element

F.7.35.3 getConfigurableElementValueExpresssion

Description: Returns the expression of a given configurable element (default expression)

- Returns: `expression` of type ***String*** - Default configurable element expression
- Input: `configurableElementID` of type ***String*** - Handle to a configurable element

F.7.35.4 getConfigurableElementValueIDs

Description: Returns all configurableElementValueIDs of the given configured element

- Returns: `configurableElementValueIDs` of type ***String[]*** - List of configurable element value handles
- Input: `configuredElementID` of type ***String*** - Handle to a configured element

F.7.35.5 getConfigurableElementValueValue

Description: Returns the value of a given configurable element value (actual value)

- Returns: `value` of type ***String*** - Value of the configurable element value
- Input: `configurableElementValueID` of type ***String*** - Handle to a configurable element value

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.35.6 getConfigurableElementValueValueExpresssion

Description: Returns the expression of a given configurable element value (actual expression)

- Returns: `expression` of type ***String*** - Expression of the configurable element value
- Input: `configurableElementValueID` of type ***String*** - Handle to a configurable element value

F.7.35.7 getConfigurableElementValueReferenceID

Description: Returns the referenceID of a given configurable element value

- Returns: `referenceID` of type ***String*** - ReferenceID of the configurable element value
- Input: `configurableElementValueID` of type ***String*** - Handle to a configurable element value

F.7.35.8 getUnconfiguredID

Description: Returns the unconfiguredID of a given configured element

- Returns: `unconfiguredElementID` of type ***String*** - Handle to a unconfigured element
- Input: `configuredElementID` of type ***String*** - Handle to a configured element

F.7.35.9 setConfigurableElementValue

Description: Set the value of a given configurable element (default value)

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `configurableElementID` of type ***String*** - Handle to a configurable element
- Input: `expression` of type ***String*** - New expression

F.7.35.10 addConfigurableElementValue

Description: Add a configurable element value with the given expression to the given configured element and the given referenceID

- Returns: `configurableElementValueID` of type ***String*** - Handle to new configurableElementValue
- Input: `configuredElementID` of type ***String*** - Handle to a configured element
- Input: `referenceID` of type ***String*** - Reference to a configurable element
- Input: `expression` of type ***String*** - New expression

F.7.35.11 removeConfigurableElementValue

Description: Remove the given configurable element value

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `configurableElementValueID` of type ***String*** - Handle to a configurable element value

F.7.35.12 setConfigurableElementValueValue

Description: Set the given expression as value for the given configurable element value

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

- Input: `configurableElementValueID` of type ***String*** - Handle to a configurable element value
- Input: `expression` of type ***String*** - New expression

F.7.35.13 setConfigurableElementValueReferenceID

Description: Set the given referenceID for the given configurable element value

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `configurableElementValueID` of type ***String*** - Handle to a configurable element value
- Input: `referenceID` of type ***String*** - New referenceID

F.7.36 Constraint set operations (BASE)

F.7.36.1 getConstraintSetReferenceName

Description: Returns the value of the constraintSetId attribute for the given constraintSet

- Returns: `referenceName` of type ***String*** - ConstraintSetId attribute value
- Input: `contraintSetID` of type ***String*** - Handle to a constraintSet element

F.7.36.2 getConstraintSetRange

Description: Returns the range for the given constraintSet

- Returns: `range` of type ***String[]*** - Range with left in index 0 and right in index 1
- Input: `contraintSetID` of type ***String*** - Handle to a constraintSet element

F.7.36.3 getConstraintSetDriveConstraints

Description: Returns the driveConstraintIDs for a given constraintSet

- Returns: `driveConstraintIDs` of type ***String[]*** - List of driveConstraint handles
- Input: `contraintSetID` of type ***String*** - Handle to a constraintSet element

F.7.36.4 getDriveConstraintType

Description: Returns the type for a given driveConstraint element: function class

- Returns: `value` of type ***String*** - Drive constraint type
- Input: `driveConstraintID` of type ***String*** - Handle to a driveConstraint element

F.7.36.5 getDriveConstraintValue

Description: Returns the value for a given driveConstraint element Format depends on the constraint type

- Returns: `value` of type ***String*** - Drive constraint value
- Input: `driveConstraintID` of type ***String*** - Handle to a driveConstraint element

F.7.36.6 getConstraintSetLoadConstraints

Description: Returns the loadConstraintIDs for a given constraintSet

- Returns: `loadConstraintIDs` of type ***String[]*** - List of loadConstraint handles
- Input: `contraintSetID` of type ***String*** - Handle to a constraintSet element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.36.7 getLoadContraintType

Description: Returns the type for a given loadConstraint element: function class

- Returns: `value` of type ***String*** - Load constraint type
- Input: `loadConstraintID` of type ***String*** - Handle to a loadConstraint element

F.7.36.8 getLoadContraintValue

Description: Returns the value for a given loadConstraint element Format is cell function and strength or cell class and strength

- Returns: `value` of type ***String*** - Load constraint value
- Input: `loadConstraintID` of type ***String*** - Handle to a loadConstraint element

F.7.36.9 getLoadContraintCount

Description: Returns the count for a given loadConstraint element

- Returns: `value` of type ***Number*** - Load constraint count
- Input: `loadConstraintID` of type ***String*** - Handle to a loadConstraint element

F.7.36.10 getConstraintSetTimingConstraints

Description: Returns the timingConstraintIDs for a given constraintSet

- Returns: `timingConstraintIDs` of type ***String[]*** - List of timingConstraint handles
- Input: `contraintSetID` of type ***String*** - Handle to a constraintSet element

F.7.36.11 getTimingConstraintClockDetails

Description: Returns the clockDetails for a given timingConstraint indicating the clock name, clock edge, and delay type

- Returns: `clockDetails` of type ***String[]*** - Clock details with clock name, clock edge, and delay type
- Input: `timingConstraintID` of type ***String*** - Handle to a timingConstraint element

F.7.36.12 getTimingConstraintValue

Description: Returns the value for a given timingConstraint element indicating the cycle time percentage

- Returns: `value` of type ***Number*** - Timing constraint value in cycle time percentage
- Input: `timingConstraintID` of type ***String*** - Handle to a timingConstraint element

F.7.37 Constraint set operations (EXTENDED)

F.7.37.1 setConstraintSetReferenceName

Description: Set the given referenceName for the constraintSetId attribute for the given constraint set

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `contraintSetID` of type ***String*** - Handle to a constraintSet element
- Input: `referenceName` of type ***String*** - ConstraintSetId attribute value

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.37.2 setConstraintSetRange

Description: Set the given range for the given constraintSet

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `contraintSetID` of type ***String*** - Handle to a constraintSet element
- Input: `range` of type ***String[]*** - Range with left at index 0 and right at index 1

F.7.37.3 addConstraintSetDriveConstraint

Description: Add a new driveConstraint with the given type to the given constraintSet

- Returns: `driveConstraintID` of type ***String*** - Handle to new driveConstraint
- Input: `contraintSetID` of type ***String*** - Handle to a constraintSet element
- Input: `type` of type ***String*** - DriveConstraint type

F.7.37.4 removeConstraintSetDriveConstraint

Description: Remove the given driveConstraint

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `driveConstraintID` of type ***String*** - Handle to a driveConstraint element

F.7.37.5 setDriveContraintValue

Description: Set the given value for the given driveConstraint element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `driveConstraintID` of type ***String*** - Handle to a driveConstraint element
- Input: `value` of type ***String*** - DriveConstraint value

F.7.37.6 addConstraintSetLoadConstraint

Description: Add a new loadConstraint with the given type to the given constraintSet

- Returns: `loadConstraintID` of type ***String*** - Handle to new loadConstraint
- Input: `contraintSetID` of type ***String*** - Handle to a constraintSet element
- Input: `type` of type ***String*** - LoadConstraint type

F.7.37.7 removeConstraintSetLoadConstraint

Description: Remove the given loadConstraint

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `loadConstraintID` of type ***String*** - Handle to a loadConstraint element

F.7.37.8 setLoadContraintValue

Description: Set the given value for the given loadConstraint element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `loadConstraintID` of type ***String*** - Handle to a loadConstraint element
- Input: `value` of type ***String*** - LoadConstraint value

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.37.9 setLoadContraintCount

Description: Set the given count for the given loadConstraint element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `loadConstraintID` of type ***String*** - Handle to a loadConstraint element
- Input: `count` of type ***String*** - LoadConstraint count

F.7.37.10 addConstraintSetTimingConstraint

Description: Add a new timingConstraint with the given clockName to the given constraintSet

- Returns: `timingConstraintID` of type ***String*** - Handle to new timingConstraint
- Input: `contraintSetID` of type ***String*** - Handle to a constraintSet element
- Input: `clockName` of type ***String*** - timingConstraint clock name

F.7.37.11 removeConstraintSetTimingConstraint

Description: Remove the given timingConstraint

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `timingConstraintID` of type ***String*** - Handle to a timingConstraint element

F.7.37.12 setTimingConstraintValue

Description: Set the given value for the given timingConstraint

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `timingConstraintID` of type ***String*** - Handle to a timingConstraint element
- Input: `value` of type ***String*** - timingConstraint value

F.7.37.13 setTimingConstraintClockEdge

Description: Set the given clockEdge for the given timingConstraint

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `timingConstraintID` of type ***String*** - Handle to a timingConstraint element
- Input: `clockEdge` of type ***String*** - timingConstraint clock edge

F.7.37.14 setTimingConstraintDelayType

Description: Set the given delayType for the given timingConstraint

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `timingConstraintID` of type ***String*** - Handle to a timingConstraint element
- Input: `delayType` of type ***String*** - timingConstraint delay type

F.7.38 Cpu operations (BASE)

F.7.38.1 getCpuAddressSpaceIDs

Description: Returns addressSpaceIDs for the given cpu element

- Returns: `addressSpaceIDs` of type ***String[]*** - List of addressSpace handles
- Input: `cpuID` of type ***String*** - Handle to a cpu element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.39 Cpu operations (EXTENDED)

F.7.39.1 addCpuAddressSpace

Description: Add the given addressSpace name to the given cpu element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `cpuID` of type ***String*** - Handle to a cpu element
- Input: `addressSpaceRef` of type ***String*** - Name of an addressSpace

F.7.40 Design configuration instantiation operations (BASE)

F.7.40.1 getDesignConfigurationInstantiationLanguage

Description: Returns language for the given designConfigurationInstantiation element

- Returns: `language` of type ***String*** - DesignConfigurationInstantiation language
- Input: `designConfigurationInstantiationID` of type ***String*** - Handle to a designConfigurationInstantiation element

F.7.40.2 getDesignConfigurationInstantiationDesignConfigurationInstanceID

Description: Returns a designConfigurationInstanceID for the given designConfigurationInstantiation element

- Returns: `designInstanceID` of type ***String*** - Handle to a designConfigurationInstance
- Input: `designConfigurationInstantiationID` of type ***String*** - Handle to a designConfigurationInstantiation element

F.7.41 Design configuration instantiation operations (EXTENDED)

F.7.41.1 setDesignConfigurationInstantiationLanguage

Description: Set language with the given value for the given designConfigurationInstantiation element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `designConfigurationInstantiationID` of type ***String*** - Handle to a designConfigurationInstantiation element
- Input: `language` of type ***String*** - DesignConfigurationInstantiation language

F.7.42 Design configuration operations (BASE)

F.7.42.1 getDesignConfigurationDesignRef

Description: Returns designVLNV for the given designConfiguration element

- Returns: `designVLNV` of type ***String[]*** - Design VLNV
- Input: `designConfigurationID` | `designConfigurationInstanceID` of type String - ***Handle*** to a designConfiguration or designConfigurationInstance element

F.7.42.2 getDesignConfigurationGeneratorChainConfigurationIDs

Description: Returns generatorChainConfigurationIDs for the given designConfiguration element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

— Returns: `generatorChainConfigurationIDs` of type ***String[]*** - List of generatorChainConfiguration handles
— Input: `designConfigurationID` | `designConfigurationInstanceID` of type ***String*** - Handle to a designConfiguration or designConfigurationInstance element

F.7.42.3 getDesignConfigurationInterconnectionConfigurationIDs

Description: Returns interconnectionConfigurationIDs for the given designConfiguration element

— Returns: `interconnectionConfigurationIDs` of type ***String[]*** - List of interconnectionConfiguration handles
— Input: `designConfigurationID` | `designConfigurationInstanceID` of type ***String*** - Handle to a designConfiguration or designConfigurationInstance element

F.7.42.4 getInterconnectionConfigurationInterconnectionID

Description: Returns interconnectionID for the given interconnectionConfiguration element

— Returns: `interconnectionID` of type ***String*** - Handle to an interconnection
— Input: `interconnectionConfigurationID` of type ***String*** - Handle to an interconnectConfiguration element

F.7.42.5 getInterconnectionConfigurationAbstractorInstantiationsIDs

Description: Returns abstractorInstancesIDs for the given interconnectionConfiguration element

— Returns: `abstractorInstancesIDs` of type ***String[]*** - List of abstractorInstances handles
— Input: `interconnectionConfigurationID` of type ***String*** - Handle to an interconnectConfiguration element

F.7.42.6 getAbstractorInstancesInterfaceComponentInstanceReference

Description: Returns componentRef for the given abstractorInstancesInterface element

— Returns: `componentInstanceName` of type ***String*** - ComponentInstance name
— Input: `abstractorInstancesInterfaceID` of type ***String*** - Handle to an abstractorInstancesInterface element

F.7.42.7 getAbstractorInstancesInterfaceBusInterfaceReference

Description: Returns busRef for the given abstractorInstancesInterface element

— Returns: `busInterfaceIName` of type ***String*** - BusInterface name
— Input: `abstractorInstancesInterfaceID` of type ***String*** - Handle to an abstractorInstancesInterface element

F.7.42.8 getAbstractorInstancesAbstractorInstanceIDs

Description: Returns the abstractorInstanceIDs for the given abstractorInstances element

— Returns: `abstractorInstanceIDs` of type ***String[]*** - List of abstractorInstance handles
— Input: `abstractorInstancesID` of type ***String*** - Handle to an abstractorInstances element

F.7.42.9 getAbstractorInstanceName

Description: Returns instanceName for the given abstractorInstance element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

- Returns: `instanceName` of type ***String*** - Abstractor instance name
- Input: `abstractorInstanceID` of type ***String*** - Handle to an abstractorInstanceElement

F.7.42.10 getAbstractorInstanceViewName

Description: Returns viewName for the given abstractorInstance element

- Returns: `viewName` of type ***String*** - Abstractor view name
- Input: `abstractorInstanceID` of type ***String*** - Handle to an abstractorInstanceElement

F.7.42.11 getDesignConfigurationViewConfigurationIDs

Description: Returns viewConfigurationIDs for the given designConfiguration element

- Returns: `viewConfigurationIDs` of type ***String[]*** - List of viewConfiguration handles
- Input: `designConfigurationID` | `designConfigurationInstanceID` of type ***String*** - Handle to a designConfiguration or designConfigurationInstance element

F.7.42.12 getViewConfigurationInstanceName

Description: Returns instanceName for the given viewConfiguration element

- Returns: `instanceName` of type ***String*** - Component instance name
- Input: `viewConfigurationID` of type ***String*** - Handle to a viewConfiguration element

F.7.42.13 getViewConfigurationViewID

Description: Returns configured viewID for the given viewConfiguration element

- Returns: `viewID` of type ***String*** - Handle to a view (configured)
- Input: `viewConfigurationID` of type ***String*** - Handle to a viewConfiguration element

F.7.42.14 createDesignConfiguration

Description: Create a new designConfiguration with the given VLNV and returns its designID; fails and returns null if VLNV already exists

- Returns: `designConfigurationID` of type ***String*** - Handle to a designConfiguration element
- Input: `designConfigurationVLNV` of type ***String[]*** - DesignConfiguration VLNV

F.7.42.15 setDesignConfigurationDesignRef

Description: Set designRef with the given VLNV for the given designConfiguration element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `designConfigurationID` of type ***String*** - Handle to a designConfiguration element
- Input: `designVLNV` of type ***String*** - Design VLNV

F.7.42.16 addDesignConfigurationGeneratorChainConfiguration

Description: Add generatorChainConfiguration with the given VLNV to the given designConfiguration element

- Returns: `generatorChainConfigurationID` of type ***String*** - Handle to a generatorChain-Configuration element
- Input: `designConfigurationID` of type ***String*** - Handle to a designConfiguration element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

— Input: `generatorVLNV` of type ***String*** - Generator VLNV

F.7.42.17 removeDesignConfigurationGeneratorChainConfiguration

Description: Remove the given generatorChainConfiguration element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

— Input: `generatorChainConfigurationID` of type ***String*** - Handle to a generatorChainConfigurationElement

F.7.42.18 addDesignConfigurationInterconnectionConfiguration

Description: Add interconnectionConfiguration with the given interconnectionRef, VLNV, instance, and viewName to the given designConfiguration element

— Returns: `interconnectionConfigurationID` of type ***String*** - Handle to an interconnectionConfiguration

— Input: `designConfigurationID` of type ***String*** - Handle to a designConfiguration element

— Input: `interconnectionRef` of type ***String*** - Interconnection name

— Input: `abstractorVLNV` of type ***String*** - Abstractor VLNV

— Input: `instanceName` of type ***String*** - Abstractor instance name

— Input: `viewName` of type ***String*** - Abstractor instance view name

F.7.42.19 removeDesignConfigurationInterconnectionConfiguration

Description: Remove the given interconnectionConfiguration element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

— Input: `interconnectionConfigurationID` of type ***String*** - Handle to an interconnectConfiguration element

F.7.42.20 addInterconnectionConfigurationAbstractorInstances

Description: Add abstractorInstances with the given VLNV, instance, and viewName to the given interconnectionConfiguration element

— Returns: `abstractorInstancesID` of type ***String*** - Handle to an abstractorInstances element

— Input: `interconnectionConfigurationID` of type ***String*** - Handle to an interconnectConfiguration element

— Input: `abstractorVLNV` of type ***String*** - Abstractor VLNV

— Input: `instanceName` of type ***String*** - Abstractor instance name

— Input: `viewName` of type ***String*** - Abstractor instance view name

F.7.42.21 removeInterconnectionConfigurationAbstractorInstances

Description: Remove the given abstractorInstances element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

— Input: `abstractorInstancesID` of type ***String*** - Handle to an abstractorInstances element

F.7.42.22 addAbstractorInstancesAbstractorInstance

Description: Add abstractorInstance with the given VLNV, instance, and viewName to the given abstractorInstances element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

- Returns: `abstractorInstanceID` of type ***String*** - Handle to an abstractorInstantiations element
- Input: `abstractorInstancesID` of type ***String*** - Handle to an abstractorInstances element
- Input: `abstractorVLNV` of type ***String*** - Abstractor VLNV
- Input: `instanceName` of type ***String*** - Abstractor instance name
- Input: `viewName` of type ***String*** - Abstractor instance view name

F.7.42.23 removeAbstractorInstancesAbstractorInstance

Description: Remove the given abstractorInstance element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `abstractorInstancesID` of type ***String*** - Handle to an abstractorInstances element

F.7.42.24 addDesignConfigurationViewConfiguration

Description: Add viewConfiguration with the given instanceName and viewName to the given designConfiguration element

- Returns: `viewConfigurationID` of type ***String*** - Handle to a viewConfiguration element
- Input: `designConfigurationID` of type ***String*** - Handle to a designConfiguration element
- Input: `componentInstanceName` of type ***String*** - ComponentInstance name
- Input: `viewName` of type ***String*** - Component view name

F.7.42.25 removeDesignConfigurationViewConfiguration

Description: Remove the given viewConfiguration element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `viewConfigurationID` of type ***String*** - Handle to a viewConfiguration element

F.7.43 Design configuration operations (EXTENDED)

F.7.43.1 getAbstractorInstancesInterfaceIDs

Description: Returns the abstractorInstancesInterfaceIDs for the given abstractorInstances element

- Returns: `abstractorInstancesInterfaceIDs` of type ***String*** - List of abstractorInstancesInterface element handles
- Input: `abstractorInstancesID` of type ***String*** - Handle to an abstractorInstances element

F.7.43.2 getAbstractorInstanceID

Description: Returns the abstractorInstanceID for the abstractorInstance element

- Returns: `abstractorInstanceID` of type ***String*** - Abstractor instance handle
- Input: `abstractorInstanceID` of type ***String*** - Handle to an abstractorInstanceElement

F.7.43.3 addAbstractorInstancesInterface

Description: Add abstractorInstancesInterface with the given componentRef and busRef to the given abstractorInstances element

- Returns: `abstractorInstancesInterfaceID` of type ***String*** - Handle to an abstractorInstantiationsInterface element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

— Input: `abstractorInstancesID` of type ***String*** - Handle to an abstractorInstances element
— Input: `componentInstanceName` of type ***String*** - ComponentInstance name
— Input: `busRef` of type ***String*** - BusInterface name

F.7.43.4 removeAbstractorInstancesInterface

Description: Remove the given abstractorInstancesInterface element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `abstractorInstancesInterfaceID` of type ***String*** - Handle to an abstractorInstancesInterface element

F.7.44 Design instantiation operations (BASE)

F.7.44.1 getDesignInstantiationDesignInstanceID

Description: Returns a designInstanceID for the given designInstantiation element

— Returns: `designInstanceID` of type ***String*** - Handle to a designInstance
— Input: `designInstantiationID` of type ***String*** - Handle to a designInstantiation element

F.7.45 Design operations (BASE)

F.7.45.1 getDesignComponentInstanceIDs

Description: Returns componentInstanceIDs for the given design element

— Returns: `componentInstanceIDs` of type ***String[]*** - List of componentInstance handles
— Input: `designID` | `designInstanceID` of type ***String*** - Handle to a design or designInstance element

F.7.45.2 getComponentInstanceName

Description: Returns name for the given componentInstance element

— Returns: `name` of type ***String*** - ComponentInstance name
— Input: `componentInstanceID` of type ***String*** - Handle to a componentInstance element

F.7.45.3 getDesignInterconnectionIDs

Description: Returns interconnectionsIDs for the given design element

— Returns: `interconnectionIDs` of type ***String[]*** - List of interconnection handles
— Input: `designID` | `designInstanceID` of type ***String*** - Handle to a design or designInstance element

F.7.45.4 getInterconnectionActiveInterfaceIDs

Description: Returns activeInterfaceIDs for the given interconnection element

— Returns: `activeInterfaceIDs` of type ***String[]*** - List of activeInterface handles
— Input: `interconnectionID` of type ***String*** - Handle to an interconnection element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.45.5 getActiveInterfaceComponentInstanceReference

Description: Returns componentRef for the given activeInterface element

- Returns: `componentInstanceName` of type ***String*** - ComponentInstance name
- Input: `activeInterfaceID` of type ***String*** - Handle to an activeInterface element

F.7.45.6 getActiveInterfaceBusInterfaceReference

Description: Returns busRef for the given activeInterface element

- Returns: `busInterfaceName` of type ***String*** - BusInterface name of component instance
- Input: `activeInterfaceID` of type ***String*** - Handle to an activeInterface element

F.7.45.7 getActiveInterfaceExcludePorts

Description: Returns excludePorts for the given activeInterface element

- Returns: `excludePortNames` of type ***String[]*** - List of port names to be excluded from connecting
- Input: `activeInterfaceID` of type ***String*** - Handle to an activeInterface element

F.7.45.8 getInterconnectionHierInterfaceIDs

Description: Returns hierInterfaceIDs for the given activeInterface element

- Returns: `hierInterfaceIDs` of type ***String[]*** - List of hierInterface handles
- Input: `interconnectionID` of type ***String*** - Handle to an interconnection element

F.7.45.9 getHierInterfaceBusInterfaceReference

Description: Returns busRef for the given hierInterface element

- Returns: `busInterfaceName` of type ***String*** - Bus interface name of top component
- Input: `hierInterfaceID` of type ***String*** - Handle to an hierInterface element

F.7.45.10 getDesignMonitorInterconnectionIDs

Description: Returns monitorInterconnectionIDs for the given design element

- Returns: `monitorInterconnectionIDs` of type ***String[]*** - List of monitorInterconnection handles
- Input: `designID` | `designInstanceID` of type ***String*** - Handle to a design or designInstance element

F.7.45.11 getMonitorInterconnectionMonitoredActiveInterfaceID

Description: Returns monitorActiveInterfaceID for the given monitorInterconnection element

- Returns: `monitoredInterfaceID` of type ***String*** - Handle to a monitoredInterface
- Input: `monitorInterconnectionID` of type ***String*** - Handle to a monitorInterconnection element

F.7.45.12 getMonitoredActiveInterfaceComponentInstanceReference

Description: Returns componentRef for the given monitoredActiveInterface element

- Returns: `componentInstanceName` of type ***String*** - ComponentInstance name

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

— Input: `monitorInterfaceID` of type ***String*** - Handle to a monitorInterface element

F.7.45.13 getMonitoredActiveInterfaceBusInterfaceReference

Description: Returns busRef for the given monitoredActiveInterface element

— Returns: `busInterfaceName` of type ***String*** - BusInterface name

— Input: `monitorInterfaceID` of type ***String*** - Handle to a monitorInterface element

F.7.45.14 getMonitoredActiveInterfacePath

Description: Returns path for the given monitoredActiveInterface element

— Returns: `path` of type ***String*** - Hierarchical path separated by slash character

— Input: `monitorInterfaceID` of type ***String*** - Handle to a monitorInterface element

F.7.45.15 getMonitorInterconnectionMonitorInterfaceIDs

Description: Returns monitorInterfaceIDs for the given monitorInterconnection element

— Returns: `monitorInterfaceIDs` of type ***String[]*** - List of monitorInterface handles

— Input: `monitorInterconnectionID` of type ***String*** - Handle to a monitorInterconnection element

F.7.45.16 getDesignAdHocConnectionIDs

Description: Returns adHocConnectionsIDs for the given design element

— Returns: `adHocConnectionIDs` of type ***String[]*** - List of adHocConnection handles

— Input: `designID` | `designInstanceID` of type ***String*** - Handle to a design or designInstance element

F.7.45.17 getAdHocConnectionTiedValue

Description: Returns tiedValue for the given adHocConnection element

— Returns: `value` of type ***String*** - AdHoc connection tied value

— Input: `adHocConnectionID` of type ***String*** - Handle to an adHocConnection element

F.7.45.18 getAdHocConnectionInternalPortReferenceIDs

Description: Returns internalPortReferenceIDs for the given adHocConnection element

— Returns: `adHocInternalPortReferenceIDs` of type ***String[]*** - List of adHocInternalPortReference handles

— Input: `adHocConnectionID` of type ***String*** - Handle to an adHocConnection element

F.7.45.19 getAdHocInternalPortReferenceComponentInstanceReference

Description: Returns componentRef for the given internalPortReference element

— Returns: `componentInstanceName` of type ***String*** - ComponentInstance name

— Input: `adHocInternalPortReferenceID` of type ***String*** - Handle to an adHocInternalPortReference element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.45.20 getAdHocInternalPortReferencePortReference

Description: Returns portRef for the given internalPortReference element

- Returns: `portName` of type ***String*** - Port name
- Input: `adHocInternalPortReferenceID` of type ***String*** - Handle to an adHocInternalPortReference element

F.7.45.21 getAdHocInternalPortReferenceRange

Description: Returns range for the given internalPortReference element

- Returns: `range` of type ***Number[]*** - Range with left at index 0 and right at index 1
- Input: `adHocInternalPortReferenceID` of type ***String*** - Handle to an adHocInternalPortReference element

F.7.45.22 getAdHocInternalPortReferenceIndices

Description: Returns indices for the given internalPortReference element

- Returns: `indices` of type ***Number[]*** - List of indices
- Input: `adHocInternalPortReferenceID` of type ***String*** - Handle to an adHocInternalPortReference element

F.7.45.23 getAdHocConnectionExternalPortReferenceIDs

Description: Returns externalPortReferenceIDs for the given adHocConnection element

- Returns: `adHocExernalPortReferenceIDs` of type ***String[]*** - List of adHocExternalPortReference handles
- Input: `adHocConnectionID` of type ***String*** - Handle to an adHocConnection element

F.7.45.24 getAdHocExternalPortReferencePortReference

Description: Returns componentRef for the given externalPortReference element

- Returns: `portName` of type ***String*** - Port name
- Input: `adHocExternalPortReferenceID` of type ***String*** - Handle to an adHocExternalPortReference element

F.7.45.25 getAdHocExternalPortReferenceRange

Description: Returns portRef for the given externalPortReference element

- Returns: `range` of type ***Number[]*** - Range with left at index 0 and right at index 1
- Input: `adHocExternalPortReferenceID` of type ***String*** - Handle to an adHocExternalPortReference element

F.7.45.26 getAdHocExternalPortReferenceIndices

Description: Returns range for the given externalPortReference element

- Returns: `indices` of type ***Number[]*** - List of indices
- Input: `adHocExternalPortReferenceID` of type ***String*** - Handle to an adHocExternalPortReference element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.45.27 createDesign

Description: Create a new design with the given VLNV and returns its designID; fails and returns null if VLNV already exists

- Returns: `designID` of type ***String*** - Handle to a design element
- Input: `designVLNV` of type ***String[]*** - Design VLNV

F.7.45.28 addDesignComponentInstance

Description: Add componentInstance with the given VLNV and instance name to the given design element

- Returns: `componentInstanceID` of type ***String*** - Handle to a component instance element
- Input: `designID` of type ***String*** - Handle to a design element
- Input: `componentVLNV` of type ***String[]*** - Component VLNV
- Input: `componentInstanceName` of type ***String*** - Component instance name

F.7.45.29 removeDesignComponentInstance

Description: Remove the given componentInstance element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `componentInstanceID` of type ***String*** - Handle to a componentInstance element

F.7.45.30 addDesignInterconnection

Description: Add interconnection with the given name, componentInstanceRef, and busRef to the given design element

- Returns: `interconnectionID` of type ***String*** - Handle to an interconnection element
- Input: `designID` of type ***String*** - Handle to a design element
- Input: `name` of type ***String*** - Interconnection name
- Input: `componentInstanceRef` of type ***String*** - Component instance name
- Input: `busInterfaceRef` of type ***String*** - Bus interface name

F.7.45.31 removeDesignInterconnection

Description: Remove the given interconnection element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `interconnectionID` of type ***String*** - Handle to an interconnection element

F.7.45.32 addInterconnectionActiveInterface

Description: Add activeInterface with the given componentInstanceRef and busRef tot the given interconnection element

- Returns: `activeInterfaceID` of type ***String*** - Handle to an activeInterface element
- Input: `interconnectionID` of type ***String*** - Handle to an interconnection element
- Input: `componentInstanceRef` of type ***String*** - Component instance name
- Input: `busInterfaceRef` of type ***String*** - Bus interface name

F.7.45.33 removeInterconnectionActiveInterface

Description: Remove the given activeInterface element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `activeInterfaceID` of type ***String*** - Handle to an activeInterface element

F.7.45.34 setActiveInterfaceExcludePorts

Description: Set excludePorts with the given value for the given activeInterface element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `activeInterfaceID` of type ***String*** - Handle to an activeInterface element
- Input: `excludePortNames` of type ***String[]*** - List of port names to be excluded from connecting

F.7.45.35 addInterconnectionHierInterface

Description: Add hierInterface with the given busRef to the given interconnection element

- Returns: `hierInterfaceID` of type ***String*** - Handle to an hierInterface element
- Input: `interconnectionID` of type ***String*** - Handle to an interconnection element
- Input: `busInterfaceRef` of type ***String*** - Bus interface name

F.7.45.36 removeInterconnectionHierInterface

Description: Remove the given hierInterface element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `hierInterfaceID` of type ***String*** - Handle to an hierInterface element

F.7.45.37 addDesignMonitorConnection

Description: Add monitorInterconnection with the given name, componentInstanceRef, and busRef to the given design element

- Returns: `monitorInterconnectionID` of type ***String*** - Handle to a monitorInterconnection element
- Input: `designID` of type ***String*** - Handle to a design element
- Input: `name` of type ***String*** - MonitorInterconnection name
- Input: `componentInstanceRef` of type ***String*** - Component instance name
- Input: `busInterfaceRef` of type ***String*** - Bus interface name

F.7.45.38 removeDesignMonitorConnection

Description: Remove the given monitorInterconnection element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `monitorInterconnectionID` of type ***String*** - Handle to a monitorInterconnection element

F.7.45.39 addMonitorInterconnectionMonitorInterface

Description: Add monitorInterface with the given componentInstanceRef and busRef to the given monitorInterconnection element

- Returns: `monitorInterfaceID` of type ***String*** - Handle to a monitorInterface element
- Input: `monitorInterconnectionID` of type ***String*** - Handle to a monitorInterconnection element
- Input: `componentInstanceRef` of type ***String*** - Component instance name

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

— Input: `busInterfaceRef` of type ***String*** - Bus interface name

F.7.45.40 removeMonitorInterconnectionMonitorInterface

Description: Remove the given monitorInterface interface

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

— Input: `monitorInterfaceID` of type ***String*** - Handle to a monitorInterface element

F.7.45.41 addDesignAdHocConnection

Description: Add adHocConnection with the given name, componentInstanceRef, and portRef to the given design element

— Returns: `adHocConnectionID` of type ***String*** - Handle to an adHocConnection element

— Input: `designID` of type ***String*** - Handle to a design element

— Input: `name` of type ***String*** - AdHocConnection name

— Input: `componentInstanceRef` of type ***String*** - Component instance name

— Input: `portRef` of type ***String*** - Port name

F.7.45.42 addDesignExternalAdHocConnection

Description: Add adHocConnection with the given name and portRef to the given design element

— Returns: `adHocConnectionID` of type ***String*** - Handle to an adHocConnection element

— Input: `designID` of type ***String*** - Handle to a design element

— Input: `name` of type ***String*** - AdHocConnection name

— Input: `portRef` of type ***String*** - Port name

F.7.45.43 removeDesignAdHocConnection

Description: Remove the given adHocConnection element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

— Input: `adHocConnectionID` of type ***String*** - Handle to an adHocConnection element

F.7.45.44 setAdHocConnectionTiedValue

Description: Set tiedValue with the given value for the given adHocConnection element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

— Input: `adHocConnectionID` of type ***String*** - Handle to an adHocConnection element

— Input: `tiedValue` of type ***String*** - AdHocConnection tied value

F.7.45.45 addAdHocConnectionInternalPortReference

Description: Add internalPortReference with the given componentInstanceRef and portRef to the given adHocConnection element

— Returns: `adHocInternalPortReferenceID` of type ***String*** - Handle to an internalPortReference element

— Input: `adHocConnectionID` of type ***String*** - Handle to an adHocConnection element

— Input: `componentInstanceRef` of type ***String*** - Component instance name

— Input: `portRef` of type ***String*** - Port name

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.45.46 removeAdHocConnectionInternalPortReference

Description: Remove the given internalPortReference element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `adHocInternalPortReferenceID` of type ***String*** - Handle to an adHocInternalPortReference element

F.7.45.47 setAdHocInternalPortReferenceRange

Description: Set range with the given range for the given internalPortReference

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `adHocInternalPortReferenceID` of type ***String*** - Handle to an adHocInternalPortReference element
- Input: `range` of type ***String[]*** - Range with left at index 0 and right at index 1

F.7.45.48 setAdHocInternalPortReferenceIndices

Description: Set indices with the given indices for the given internalPortReference

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `adHocInternalPortReferenceID` of type ***String*** - Handle to an adHocInternalPortReference element
- Input: `indices` of type ***String[]*** - List of indices

F.7.45.49 addAdHocConnectionExternalPortReference

Description: Add externalPortReference with the portRef to the given adHocConnection element

- Returns: `adHocExternalPortReferenceID` of type ***String*** - Handle to an externalPortReference element
- Input: `adHocConnectionID` of type ***String*** - Handle to an adHocConnection element
- Input: `portRef` of type ***String*** - Port name

F.7.45.50 removeAdHocConnectionExternalPortReference

Description: Remove the given externalPortReference element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `adHocExternalPortReferenceID` of type ***String*** - Handle to an adHocExternalPortReference element

F.7.45.51 setAdHocExternalPortReferenceRange

Description: Set range with the given range for the given externalPortReference

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `adHocExternalPortReferenceID` of type ***String*** - Handle to an adHocExternalPortReference element
- Input: `range` of type ***String[]*** - Range with left at index 0 and right at index 1

F.7.45.52 setAdHocExternalPortReferenceIndices

Description: Set indices with the given indices for the given externalPortReference

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `adHocExternalPortReferenceID` of type ***String*** - Handle to an adHocExternalPortReference element
- Input: `indices` of type ***String[]*** - List of indices

F.7.46 Design operations (EXTENDED)

F.7.46.1 SetMonitoredActiveInterfacePath

Description: Set path with the given value for the given monitoredActiveInterface element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `monitoredActiveInterfaceID` of type ***String*** - Handle to a monitoredActiveInterface element
- Input: `path` of type ***String*** - Hierarchical path separated by slash

F.7.46.2 SetMonitorInterfacePath

Description: Set path with the given value for the given monitorInterface element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `monitorInterfaceID` of type ***String*** - Handle to a monitorInterface element
- Input: `path` of type ***String*** - Hierarchical path separated by slash

F.7.47 Driver operations (BASE)

F.7.47.1 getDriverRange

Description: Returns the range of the given driver element

- Returns: `range` of type ***String[]*** - Range with left at index 0 and right at index 1
- Input: `driverID` of type ***String*** - Handle to a driver element

F.7.47.2 getDriverDefaultValue

Description: Returns the default value of the given driver element

- Returns: `defaultValue` of type ***String*** - Default value
- Input: `driverID` of type ***String*** - Handle to a driver element

F.7.47.3 getDriverClockDriverID

Description: Returns the clockDriverID of the given driver element

- Returns: `clockDriverID` of type ***String*** - Handle to the clockDriver
- Input: `driverID` of type ***String*** - Handle to a driver element

F.7.47.4 getClockDriverClockPeriodUnits

Description: Returns the clockPeriodUnits of the given clockDriver element

- Returns: `clockPeriodUnits` of type ***String*** - Clock period units value
- Input: `clockDriverID` of type ***String*** - Handle to a clockDriver element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.47.5 getClockDriverClockPeriod

Description: Returns the clockPeriod of the given clockDriver element

- Returns: `clockPeriod` of type ***Number*** - Clock period value
- Input: `clockDriverID` of type ***String*** - Handle to a clockDriver element

F.7.47.6 getClockDriverClockPulseOffsetUnits

Description: Returns the clockPulseOffsetUnits of the given clockDriver element

- Returns: `clockPulseOffsetUnits` of type ***String*** - Clock pulse offset units value
- Input: `clockDriverID` of type ***String*** - Handle to a clockDriver element

F.7.47.7 getClockDriverClockPulseOffset

Description: Returns the clockPulseOffset of the given clockDriver element

- Returns: `clockPulseOffset` of type ***Number*** - Clock pulse offset value
- Input: `clockDriverID` of type ***String*** - Handle to a clockDriver element

F.7.47.8 getClockDriverClockPulseValue

Description: Returns the clockPulseValue of the given clockDriver element

- Returns: `clockPulseValue` of type ***Number*** - Clock pulse value
- Input: `clockDriverID` of type ***String*** - Handle to a clockDriver element

F.7.47.9 getClockDriverClockPulseDurationUnits

Description: Returns the clockPulseDurationUnits of the given clockDriver element

- Returns: `clockPulseDurationUnits` of type ***String*** - Clock pulse duration units value
- Input: `clockDriverID` of type ***String*** - Handle to a clockDriver element

F.7.47.10 getClockDriverClockPulseDuration

Description: Returns the clockPulseDuration of the given clockDriver element

- Returns: `clockPulseDuration` of type ***Number*** - Clock pulse duration value
- Input: `clockDriverID` of type ***String*** - Handle to a clockDriver element

F.7.47.11 getDriverSingleShotDriverID

Description: Returns the singleShotDriverID of the given driver element

- Returns: `singleShotDriverID` of type ***String*** - Handle to the singleShotDriver
- Input: `driverID` of type ***String*** - Handle to a driver element

F.7.47.12 getSingleShotDriverSingleShotOffset

Description: Returns the singleShotOffset of the given singleShotDriver element

- Returns: `singleShotOffset` of type ***Number*** - Single shot offset
- Input: `singleShotDriverID` of type ***String*** - Handle to a singleShotDriver element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.47.13 getSingleShotDriverSingleShotValue

Description: Returns the singleShotValue of the given singleShotDriver element

- Returns: `singleShotValue` of type ***Number*** - Single shot value
- Input: `singleShotDriverID` of type ***String*** - Handle to a singleShotDriver element

F.7.47.14 getSingleShotDriverSingleShotDuration

Description: Returns the singleShotDuration of the given singleShotDriver element

- Returns: `singleShotDuration` of type ***Number*** - Single shot duration
- Input: `singleShotDriverID` of type ***String*** - Handle to a singleShotDriver element

F.7.48 Driver operations (EXTENDED)

F.7.48.1 setDriverRange

Description: Set the range of the given driver

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `driverID` of type ***String*** - Handle to a driver element
- Input: `range` of type ***String[]*** - Range with left at index 0 and right at index 1

F.7.48.2 addDriverDefaultValue

Description: Add the given default value to the given driver element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `driverID` of type ***String*** - Handle to a driver element
- Input: `expression` of type ***String*** - Default value expression

F.7.48.3 removeDriverDefaultValue

Description: Remove the default value from the given driver element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `driverID` of type ***String*** - Handle to a driver element
- Input: `period` of type ***String*** - Clock period expression
- Input: `offset` of type ***String*** - Clock pulse offset expression
- Input: `value` of type ***String*** - Clock pulse value expression
- Input: `duration` of type ***String*** - Clock pulse duration expression

F.7.48.4 addDriverClockDriver

Description: Add a clock driver with the given details to the given driver element

- Returns: `clockDriverID` of type ***String*** - Handle to a new clockDriver
- Input: `driverID` of type ***String*** - Handle to a driver element

F.7.48.5 removeDriverClockDriver

Description: Remove the clock driver from the given driver element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

— Input: `driverID` of type ***String*** - Handle to a driver element

F.7.48.6 setClockDriverClockPeriodUnits

Description: Set the units attribute of the clockPeriod element in the given clockDriver element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `clockDriverID` of type ***String*** - Handle to a clockDriver element
— Input: `units` of type ***String*** - Units value

F.7.48.7 setClockDriverClockPulseOffsetUnits

Description: Set the units attribute of the clockPulseOffset element in the given clockDriver element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `clockDriverID` of type ***String*** - Handle to a clockDriver element
— Input: `units` of type ***String*** - Units value

F.7.48.8 setClockDriverClockPulseDurationUnits

Description: Set the units attribute of the clockPulseDuration element in the given clockDriver element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `clockDriverID` of type ***String*** - Handle to a clockDriver element
— Input: `units` of type ***String*** - Units value

F.7.48.9 addDriverSingleShotDriver

Description: Add a single shot driver with the given details to the given driver element

— Returns: `singleShotDriverID` of type ***String*** - Handle to a new singleShotDriver
— Input: `driverID` of type ***String*** - Handle to a driver element
— Input: `offset` of type ***String*** - Single shot offset expression
— Input: `value` of type ***String*** - Single shot value expression
— Input: `duration` of type ***String*** - Single shot duration expression

F.7.48.10 removeDriverSingleShotDriver

Description: Remove the single shot driver from the given driver element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `driverID` of type ***String*** - Handle to a driver element

F.7.49 Element attribute operations (BASE)

F.7.49.1 getAttributeValue

Description: Returns value of the attribute with the given name in the given element

— Returns: `value` of type ***String*** - Attribute value
— Input: `elementID` of type ***String*** - Handle to an element that has an attribute
— Input: `name` of type ***String*** - Attribute name

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.49.2 getAttributeValueExpression

Description: Returns expression of the attribute with the given name in the given element

- Returns: `expression` of type ***String*** - Attribute expression
- Input: `elementID` of type ***String*** - Handle to an element that has an attribute
- Input: `name` of type ***String*** - Attribute name

F.7.50 Element attribute operations (EXTENDED)

F.7.50.1 addAttribute

Description: Add attribute with the given name and expression to the given element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to an element that has an attribute
- Input: `name` of type ***String*** - Attribute name
- Input: `expression` of type ***String*** - Attribute expression

F.7.50.2 removeAttribute

Description: Remove attribute with the given name from the given element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to an element that has an attribute
- Input: `name` of type ***String*** - Attribute name

F.7.50.3 setAttributeValue

Description: Set attribute with the given name to the given expression in the given element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `elementID` of type ***String*** - Handle to an element that has an attribute
- Input: `name` of type ***String*** - Attribute name
- Input: `expression` of type ***String*** - Attribute expression

F.7.51 Field operations (BASE)

F.7.51.1 getRegisterFieldID

Description: Returns the value of the FieldID attribute in the given regField element

- Returns: `id` of type ***String*** - FieldID attribute value
- Input: `regFieldID` of type ***String*** - Handle to a regField element

F.7.52 File builder operations (BASE)

F.7.52.1 getFileBuilderFileType

Description: Return fileType for the given fileBuilder element

- Returns: `fileType` of type ***String*** - FileBuilder fileType
- Input: `fileBuilderID` of type ***String*** - Handle to a fileBuilder element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.52.2 getFileBuilderCommand

Description: Return command for the given fileBuilder element

- Returns: `command` of type ***String*** - FileBuilder command
- Input: `fileBuilderID` of type ***String*** - Handle to a fileBuilder element

F.7.52.3 getFileBuilderFlags

Description: Return flags for the given fileBuilder element

- Returns: `flags` of type ***String*** - FileBuilder flags
- Input: `fileBuilderID` of type ***String*** - Handle to a fileBuilder element

F.7.52.4 getFileBuilderReplaceDefaultFlags

Description: Return replaceDefaultFlags for the given fileBuilder element

- Returns: `value` of type ***Boolean*** - FileBuilder replace default flags
- Input: `fileBuilderID` of type ***String*** - Handle to a fileBuilder element

F.7.53 File builder operations (EXTENDED)

F.7.53.1 setFileBuilderCommand

Description: Set command with the given value for the given fileBuilder element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `fileBuilderID` of type ***String*** - Handle to a fileBuilder element
- Input: `command` of type ***String*** - FileBuilder command

F.7.53.2 setFileBuilderFlags

Description: Set flags with the given value for the given fileBuilder element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `fileBuilderID` of type ***String*** - Handle to a fileBuilder element
- Input: `flags` of type ***String*** - FileBuilder flags

F.7.53.3 setFileBuilderReplaceDefaultFlags

Description: Set replaceDefaultFlags with the given value for the given fileBuilder element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `fileBuilderID` of type ***String*** - Handle to a fileBuilder element
- Input: `value` of type ***Boolean*** - FileBuilder replace default flags

F.7.54 File set operations (BASE)

F.7.54.1 getFileSetGroups

Description: Returns groups for the given fileSet element

- Returns: `groups` of type ***String[]*** - FileSet groups
- Input: `fileSetID` of type ***String*** - Handle to a fileSet element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.54.2 getFileSetFileIDs

Description: Returns fileIDs for the given fileSet element

- Returns: `fileIDs` of type ***String[]*** - List of file handles
- Input: `fileSetID` of type ***String*** - Handle to a fileSet element

F.7.54.3 getFileFileTypes

Description: Returns fileTypes for the given file element

- Returns: `fileTypes` of type ***String[]*** - File fileType
- Input: `fileID` of type ***String*** - Handle to a file element

F.7.54.4 getFileIsStructural

Description: Returns isStructural for the given file element

- Returns: `value` of type ***Boolean*** - File isStructural value
- Input: `fileID` of type ***String*** - Handle to a file element

F.7.54.5 getFileIsIncludeFile

Description: Returns isIncludeFile for the given file element

- Returns: `value` of type ***Boolean*** - File inIncludeFile value
- Input: `fileID` of type ***String*** - Handle to a file element

F.7.54.6 getFileHasExternalDeclarations

Description: Returns hasExternalDeclarations for the given file element

- Returns: `value` of type ***Boolean*** - File hasExternalDeclarations value
- Input: `fileID` of type ***String*** - Handle to a file element

F.7.54.7 getFileLogicalName

Description: Returns logicalName for the given file element

- Returns: `logicalName` of type ***String*** - File logicalName value
- Input: `fileID` of type ***String*** - Handle to a file element

F.7.54.8 getFileExportedNames

Description: Returns exportedNames for the given file element

- Returns: `exportedNames` of type ***String[]*** - File exportedNames
- Input: `fileID` of type ***String*** - Handle to a file element

F.7.54.9 getFileBuildCommand

Description: Returns buildCommand for the given file element

- Returns: `command` of type ***String*** - File buildCommand
- Input: `fileID` of type ***String*** - Handle to a file element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.54.10 getFileBuildCommandFlags

Description: Returns buildCommandFlags for the given file element

- Returns: `flags` of type ***String*** - File buildCommand flags
- Input: `fileID` of type ***String*** - Handle to a file element

F.7.54.11 getFileBuildCommandReplaceDefaultFlags

Description: Returns buildCommandReplaceDefaultFlags for the given file element

- Returns: `value` of type ***Boolean*** - File buildCommands replace default flags
- Input: `fileID` of type ***String*** - Handle to a file element

F.7.54.12 getFileBuildCommandTargetName

Description: Returns buildCommandTargetName for the given file element

- Returns: `name` of type ***String*** - File buildCommands target name
- Input: `fileID` of type ***String*** - Handle to a file element

F.7.54.13 getFileDependencies

Description: Returns dependencies for the given file element

- Returns: `dependencies` of type ***String[]*** - File dependencies
- Input: `fileID` of type ***String*** - Handle to a file element

F.7.54.14 getFileDefineSymbolIDs

Description: Returns fileDefineIDs for the given file element

- Returns: `fileDefineIDs` of type ***String[]*** - List of fileDefine handles
- Input: `fileID` of type ***String*** - Handle to a file element

F.7.54.15 getFileDefineSymbolValue

Description: Returns value for the given fileDefine element

- Returns: `value` of type ***String*** - FileDefine value
- Input: `fileDefineID` of type ***String*** - Handle to a fileDefine element

F.7.54.16 getFileImageTypes

Description: Returns imageTypes for the given file element

- Returns: `imageTypes` of type ***String[]*** - File imageTypes
- Input: `fileID` of type ***String*** - Handle to a file element

F.7.54.17 getFileSetDefaultFileBuilderIDs

Description: Returns fileBuilderIDs for the given fileSet element

- Returns: `fileBuilderIDs` of type ***String[]*** - List of fileBuilder handles
- Input: `fileSetID` of type ***String*** - Handle to a fileSet element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.54.18 getFileSetDependencies

Description: Returns dependencies for the given fileSet element

- Returns: `dependencies` of type ***String[]*** - FileSet dependencies
- Input: `fileSetID` of type ***String*** - Handle to a fileSet element

F.7.54.19 getFileSetFunctionIDs

Description: Returns functionsIDs for the given fileSet element

- Returns: `functionIDs` of type ***String[]*** - List of function handles
- Input: `fileSetID` of type ***String*** - Handle to a fileSet element

F.7.54.20 getFunctionReplicate

Description: Returns replicate for the given function element

- Returns: `replicate` of type ***Boolean*** - Function replicate
- Input: `functionID` of type ***String*** - Handle to a function element

F.7.54.21 getFunctionEntryPoint

Description: Returns entryPoint for the given function element

- Returns: `entryPoint` of type ***String*** - Function entryPoint
- Input: `functionID` of type ***String*** - Handle to a function element

F.7.54.22 getFunctionFileID

Description: Returns fileID for the given function element

- Returns: `fileID` of type ***String*** - Handle to a file
- Input: `functionID` of type ***String*** - Handle to a function element

F.7.54.23 getFunctionReturnType

Description: Returns returnType the given function element

- Returns: `returnType` of type ***String*** - Function returnType
- Input: `functionID` of type ***String*** - Handle to a function element

F.7.54.24 getFunctionArgumentIDs

Description: Returns argumentIDs for the given function element

- Returns: `argumentIDs` of type ***String[]*** - List of argument handles
- Input: `functionID` of type ***String*** - Handle to a function element

F.7.54.25 getFunctionArgumentDataType

Description: Returns dataType for the given argumentID element

- Returns: `dataType` of type ***String*** - Argument data type
- Input: `argumentID` of type ***String*** - Handle to an argument element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.54.26 getFunctionDisabled

Description: Returns disabled for the given function element

- Returns: `value` of type ***Boolean*** - Function disabled
- Input: `functionID` of type ***String*** - Handle to a function element

F.7.54.27 getFunctionSourceFileIDs

Description: Returns sourceFileIDs for the given function element

- Returns: `functionSourceFileIDs` of type ***String[]*** - List of sourceFile handles
- Input: `functionID` of type ***String*** - Handle to a function element

F.7.54.28 getFunctionSourceFileName

Description: Returns sourceName for the given functionSourceFile element

- Returns: `name` of type ***String*** - SourceFile name
- Input: `functionSourceFileID` of type ***String*** - Handle to a functionSourceFile element

F.7.54.29 getFunctionSourceFileType

Description: Returns fileType for the given functionSourceFile element

- Returns: `type` of type ***String*** - SourceFile type
- Input: `functionSourceFileID` of type ***String*** - Handle to a functionSourceFile element

F.7.54.30 getFunctionSourceFileTypeUser

Description: Returns user attribute for fileType for the given functionSourceFile element

- Returns: `user` of type ***String*** - SourceFile type user
- Input: `functionSourceFileID` of type ***String*** - Handle to a functionSourceFile element

F.7.55 File set operations (EXTENDED)

F.7.55.1 setFileSetGroups

Description: Set groups with the given value for the given fileSet element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `fileSetID` of type ***String*** - Handle to a fileSet element
- Input: `groups` of type ***String[]*** - FileSet groups

F.7.55.2 addFileSetFile

Description: Add file with the given name and fileTypes to the given fileSet element

- Returns: `fileID` of type ***String*** - Handle to a new file
- Input: `fileSetID` of type ***String*** - Handle to a fileSet element
- Input: `name` of type ***String*** - File name
- Input: `fileTypes` of type ***String[]*** - File fileTypes

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.55.3 removeFileSetFile

Description: Remove the given file

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `fileID` of type ***String*** - Handle to a file element

F.7.55.4 setFileIsStructural

Description: Set isStructural with the given value for the given file element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `fileID` of type ***String*** - Handle to a file element
- Input: `value` of type ***Boolean*** - File isStructural value

F.7.55.5 setFileIsIncludeFile

Description: Set isIncludeFile with the given value for the given file element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `fileID` of type ***String*** - Handle to a file element
- Input: `value` of type ***Boolean*** - File inIncludeFile value

F.7.55.6 setFileHasExternalDeclarations

Description: Set hasExternalDeclarations with the given value for the given file element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `fileID` of type ***String*** - Handle to a file element
- Input: `value` of type ***Boolean*** - File hasExternalDeclarations value

F.7.55.7 setFileLogicalName

Description: Set logicalName with the given value for the given file element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `fileID` of type ***String*** - Handle to a file element
- Input: `logicalName` of type ***String*** - File logicalName value

F.7.55.8 setFileExportedNames

Description: Set exportedNames with the given value for the given file element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `fileID` of type ***String*** - Handle to a file element
- Input: `exportedNames` of type ***String[]*** - File exportedNames

F.7.55.9 setFileBuildCommand

Description: Set buildCommand with the given value for the given file element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `fileID` of type ***String*** - Handle to a file element
- Input: `command` of type ***String*** - File buildCommand

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.55.10 setFileBuildCommandFlags

Description: Set buildCommandFlags with the given value for the given file element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `fileID` of type ***String*** - Handle to a file element
- Input: `flags` of type ***String*** - File buildCommand flags

F.7.55.11 setFileBuildCommandReplaceDefaultFlags

Description: Set buildCommandReplaceDefaultFlags with the given value for the given file element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `fileID` of type ***String*** - Handle to a file element
- Input: `value` of type ***Boolean*** - File buildCommand replace default flags

F.7.55.12 setFileBuildCommandTargetName

Description: Set buildCommandReplaceDefaultFlags with the given value for the given file element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `fileID` of type ***String*** - Handle to a file element
- Input: `name` of type ***String*** - File buildCommand target name

F.7.55.13 addFileDependency

Description: Add given dependency to the given file element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `fileID` of type ***String*** - Handle to a file element
- Input: `dependency` of type ***String*** - File dependency

F.7.55.14 removeFileSetDependency

Description: Remove the given dependency from the given file element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `fileID` of type ***String*** - Handle to a file element
- Input: `dependency` of type ***String*** - File dependency

F.7.55.15 addFileDefineSymbol

Description: Add fileDefine with the given name and value to the given file element

- Returns: `fileDefineID` of type ***String*** - Handle to a new fileDefine
- Input: `fileID` of type ***String*** - Handle to a file element
- Input: `name` of type ***String*** - FileDefine name
- Input: `value` of type ***String*** - FileDefine value

F.7.55.16 removeFileDefineSymbol

Description: Remove the given fileDefine element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `fileDefineID` of type ***String*** - Handle to a fileDefine element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.55.17 setFileImageTypes

Description: Set imageTypes with the given value for the given file element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `fileID` of type ***String*** - Handle to a file element
- Input: `imageTypes` of type ***String[]*** - File imageTypes

F.7.55.18 addFileSetDefaultFileBuilder

Description: Add defaultFileBuilder with the given fileType to the given file element

- Returns: `fileBuilderID` of type ***String*** - Handle to a new fileBuilder
- Input: `fileID` of type ***String*** - Handle to a file element
- Input: `fileType` of type ***String*** - DefaultFileBuilder fileType

F.7.55.19 removeFileSetDefaultFileBuilder

Description: Remove the given defaultFileBuilder element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `fileBuilderID` of type ***String*** - Handle to a fileBuilder element

F.7.55.20 addFileSetDependency

Description: Add given dependency to the given fileSet element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `fileSetID` of type ***String*** - Handle to a fileSet element
- Input: `dependency` of type ***String*** - FileSet dependency

F.7.55.21 removeFileSetDependency

Description: Remove the given dependency from the given fileSet element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `fileSetID` of type ***String*** - Handle to a fileSet element
- Input: `dependency` of type ***String*** - FileSet dependency

F.7.55.22 addFileSetFunction

Description: Add function with the given fileRef to the given fileSet element

- Returns: `functionID` of type ***String*** - Handle to a new function
- Input: `fileSetID` of type ***String*** - Handle to a fileSet element
- Input: `fileRef` of type ***String*** - Reference to the file that contains the entry point for the function

F.7.55.23 removeFileSetFunction

Description: Remove the given function element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `functionID` of type ***String*** - Handle to a function element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.55.24 setFunctionReplicate

Description: Set replicate to the given value for the given function element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `functionID` of type ***String*** - Handle to a function element
- Input: `replicate` of type ***Boolean*** - Function replicate

F.7.55.25 setFunctionEntryPoint

Description: Set entryPoint to the given value for the given function element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `functionID` of type ***String*** - Handle to a function element
- Input: `entryPoint` of type ***String*** - Function entryPoint

F.7.55.26 setFunctionReturnType

Description: Set returnType to the given value for the given function element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `functionID` of type ***String*** - Handle to a function element
- Input: `returnType` of type ***String*** - Function returnType

F.7.55.27 addFunctionArgument

Description: Add argument with given name and value to the given function element

- Returns: `argumentID` of type ***String*** - Handle to a new argument
- Input: `functionID` of type ***String*** - Handle to a function element
- Input: `name` of type ***String*** - Argument name
- Input: `value` of type ***String*** - Argument value

F.7.55.28 removeFunctionArgument

Description: Remove the given argument

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `argumentID` of type ***String*** - Handle to an argument element

F.7.55.29 setFunctionArgumentDataType

Description: Set datatype with the given type for the given argument element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `argumentID` of type ***String*** - Handle to an argument element
- Input: `dataType` of type ***String*** - Argument dataType

F.7.55.30 setFunctionDisabled

Description: Set disabled to the given value for the given function element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `functionID` of type ***String*** - Handle to a function element
- Input: `disabled` of type ***Boolean*** - Function disabled

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.55.31 addFunctionSourceFile

Description: Add sourceFile with the given sourceName and fileType to the given function element

- Returns: `functionSourceFileID` of type ***String*** - Handle to a new sourceFile
- Input: `functionID` of type String - ***Handle*** to a function element
- Input: `sourceFileName` of type ***String*** - Source file name
- Input: `fileType` of type ***String*** - Source file type

F.7.55.32 removeFunctionSourceFile

Description: Remove the given sourceFile element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `functionSourceFileID` of type ***String*** - Handle to a functionSourceFile element

F.7.55.33 setFunctionSourceFileTypeUser

Description: Set user attribute for the given sourceFile element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `functionSourceFileID` of type ***String*** - Handle to a functionSourceFile element
- Input: `user` of type ***String*** - Source file type user

F.7.56 Generator chain operations (BASE)

F.7.56.1 getGeneratorChainGeneratorChainSelectorGroupSelectorIDs

Description: Returns generatorChainSelectorGroupSelectorIDs for the given generatorChain element

- Returns: `groupSelectorIDs` of type ***String[]*** - List of groupSelector handles
- Input: `generatorChainID` of type ***String*** - Handle to a generatorChain element

F.7.56.2 getGroupSelectorSelectionOperator

Description: Returns multipleGroupSelectionOperation for a given groupSelector element

- Returns: `operator` of type ***String*** - Operator expression
- Input: `groupSelectorID` of type ***String*** - Handle to a groupSelector element

F.7.56.3 getGroupSelectorSelectionNames

Description: Returns names for a given groupSelector element

- Returns: `names` of type ***String[]*** - List of generator(chain) group names
- Input: `groupSelectorID` of type ***String*** - Handle to a groupSelector element

F.7.56.4 getGeneratorChainGeneratorChainSelectorGeneratorChainIDs

Description: Returns generatorChainIDs for the given generatorChain element

- Returns: `generatorChainIDs` of type ***String[]*** - List of generatorChain handles
- Input: `generatorChainID` of type ***String*** - Handle to a generatorChain element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.56.5 getGeneratorChainComponentGeneratorSelectorGroupSelectorIDs

Description: Returns generatorChainComponentGeneratorSelectorGroupSelectorIDs for the given generatorChain element

- Returns: `groupSelectorIDs` of type ***String[]*** - List of groupSelector handles
- Input: `generatorChainID` of type ***String*** - Handle to a generatorChain element

F.7.56.6 getGeneratorChainGeneratorIDs

Description: Returns generatorIDs for the given generatorChain element

- Returns: `generatorIDs` of type ***String*** - List of generator handles
- Input: `generatorChainID` of type ***String*** - Handle to a generatorChain element

F.7.56.7 getGeneratorChainGroups

Description: Returns chainGroups for the given generatorChain element

- Returns: `groups` of type ***String[]*** - List of chainGroups
- Input: `generatorChainID` of type ***String*** - Handle to a generatorChain element

F.7.56.8 getGeneratorChainChoiceIDs

Description: Returns choiceIDs for the given generatorChain element

- Returns: `choiceIDs` of type ***String[]*** - List of choice handles
- Input: `generatorChainID` of type ***String*** - Handle to a generatorChain element

F.7.57 Generator chain operations (EXTENDED)

F.7.57.1 createGeneratorChain

Description: Create a new generatorChain with the given VLNV and returns its generatorChainID; fails and returns null if VLNV already exists

- Returns: `generatorChainID` of type ***String*** - Handle to a new generatorChain element
- Input: `generatorChainVLNV` of type ***String[]*** - GeneratorChain VLNV

F.7.57.2 addGeneratorChainGeneratorChainSelectorGroupSelector

Description: Add generatorChainMultipleGroupSelector with the given names for a given generatorChain element

- Returns: `groupSelectorID` of type ***String*** - Handle to a groupSelector element
- Input: `generatorChainID` of type ***String*** - Handle to a generatorChain element
- Input: `names` of type ***String[]*** - Non-empty list of names

F.7.57.3 setGroupSelectorSelectionOperator

Description: Set multipleGroupSelectionOperation with the given value for a given groupSelector element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `groupSelectorID` of type ***String*** - Handle to a groupSelector element
- Input: `operator` of type ***String*** - MultipleGroupSelectorOperator attribute value which can take ‘or’ or ‘and’

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.57.4 addGeneratorChainComponentGeneratorSelectorGroupSelector

Description: Add generatorChainComponentGeneratorSelectorGroupSelector with the given names for the given generatorChain element

- Returns: `groupSelectorID` of type ***String*** - Handle to a groupSelector element
- Input: `generatorChainID` of type ***String*** - Handle to a generatorChain element
- Input: `names` of type ***String[]*** - Non-empty list of names

F.7.57.5 addGeneratorChainGenerator

Description: Add generator with the given generatorExe to the given generatorChain element

- Returns: `generatorID` of type ***String*** - Handle to a generator element
- Input: `generatorChainID` of type ***String*** - Handle to a generatorChain element
- Input: `generatorExecutable` of type ***String*** - Path to generator executable

F.7.57.6 addGeneratorChainGroup

Description: Add chainGroup with the given value to the given generatorChain element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `generatorChainID` of type ***String*** - Handle to a generatorChain element
- Input: `group` of type ***String*** - ChainGroup value

F.7.57.7 removeGeneratorChainGroup

Description: Remove chainGroup with the given value from the given generatorChain element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `generatorChainID` of type ***String*** - Handle to a generatorChain element
- Input: `group` of type ***String*** - ChainGroup value

F.7.57.8 addGeneratorChainChoice

Description: Add choice with the given name and enumerations to the given generatorChain element

- Returns: `choiceID` of type ***String*** - Handle to a choice element
- Input: `generatorChainID` of type ***String*** - Handle to a generatorChain element
- Input: `name` of type ***String*** - Choice name
- Input: `enumerations` of type ***String[]*** - Choice enumeration values

F.7.57.9 removeGeneratorChainChoice

Description: Remove the given choice element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `choiceID` of type ***String*** - Handle to a choice element

F.7.58 Generator operations (BASE)

F.7.58.1 getGeneratorHidden

Description: Returns hidden for the given generator element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

- Returns: `hidden` of type ***String*** - Generator hidden
- Input: `generatorID` of type ***String*** - Handle to a generator element

F.7.58.2 getGeneratorScope

Description: Returns scope for the given generator element

- Returns: `scope` of type ***String*** - Generator scope
- Input: `generatorID` of type ***String*** - Handle to a generator element

F.7.58.3 getGeneratorPhase

Description: Returns phase for the given generator element

- Returns: `phase` of type ***Number*** - Generator phase
- Input: `generatorID` of type ***String*** - Handle to a generator element

F.7.58.4 getGeneratorApiType

Description: Returns apiType for the given generator element

- Returns: `apiType` of type ***String*** - Generator apiType
- Input: `generatorID` of type ***String*** - Handle to a generator element

F.7.58.5 getGeneratorTransportMethods

Description: Returns transportMethods for the given generator element

- Returns: `transportMethods` of type ***String[]*** - Generator transportMethods
- Input: `generatorID` of type ***String*** - Handle to a generator element

F.7.58.6 getGeneratorExecutable

Description: Returns generatorExe for the given generator element

- Returns: `executable` of type ***String*** - Generator executable
- Input: `generatorID` of type ***String*** - Handle to a generator element

F.7.58.7 getGeneratorGroups

Description: Returns group names for the given generator element

- Returns: `groups` of type ***String[]*** - Generator groups
- Input: `generatorID` of type ***String*** - Handle to a generator element

F.7.59 Generator operations (EXTENDED)

F.7.59.1 setGeneratorHidden

Description: Set hidden with the given value for the given generator element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `generatorID` of type ***String*** - Handle to a generator element
- Input: `hidden` of type ***String*** - Generator hidden

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.59.2 setGeneratorScope

Description: Set scope with the given value for the given generator element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `generatorID` of type ***String*** - Handle to a generator element
- Input: `scope` of type ***String*** - Generator scope

F.7.59.3 setGeneratorPhase

Description: Set phase with the given value for the given generator element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `generatorID` of type ***String*** - Handle to a generator element
- Input: `phase` of type ***Number*** - Generator phase

F.7.59.4 setGeneratorApiType

Description: Set apiType with the given value for the given generator element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `generatorID` of type ***String*** - Handle to a generator element
- Input: `apiType` of type ***String*** - Generator apiType

F.7.59.5 setGeneratorTransportMethods

Description: Set transportMethods with the given value for the given generator element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `generatorID` of type ***String*** - Handle to a generator element
- Input: `transportMethods` of type ***String[]*** - Generator transportMethods

F.7.59.6 setGeneratorExecutable

Description: Set generatorExe with the given value for the given generator element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `generatorID` of type ***String*** - Handle to a generator element
- Input: `executable` of type ***String*** - Generator executable

F.7.59.7 setGeneratorGroups

Description: Set groups with the given value for the given generator element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `generatorID` of type ***String*** - Handle to a generator element
- Input: `groups` of type ***String[]*** - Generator groups

F.7.60 Global operations (BASE)

F.7.60.1 getAbstractionDefIDs

Description: Returns all registered abstractionDefIDs

- Returns: `abstractionDefIDs` of type ***String[]*** - List of abstraction definition handles

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.60.2 getBusDefIDs

Description: Returns all registered busDefIDs

- Returns: `busDefIDs` of type ***String[]*** - List of bus definition handles

F.7.60.3 getComponentIDs

Description: Returns all registered componentIDs

- Returns: `componentIDs` of type ***String[]*** - List of component handles

F.7.60.4 getDesignIDs

Description: Returns all registered designIDs

- Returns: `designIDs` of type ***String[]*** - List of design handles

F.7.60.5 getDesignConfigurationIDs

Description: Returns all registered designConfigurationIDs

- Returns: `designConfigurationIDs` of type ***String[]*** - List of design configuration handles

F.7.60.6 getAbstractorIDs

Description: Returns all registered abstractorIDs

- Returns: `abstractorIDs` of type ***String[]*** - List of abstractor handles

F.7.60.7 getGeneratorChainIDs

Description: Returns all registered generatorChainIDs

- Returns: `generatorChainIDs` of type ***String[]*** - List of generator chain handles

F.7.60.8 getCatalogIDs

Description: Returns all registered catalogIDs

- Returns: `catalogIDs` of type ***String[]*** - List of catalog handles

F.7.60.9 getDesignID

Description: Returns the designID of the current or top design associated with the currently invoked generator

- Returns: `designID` of type ***String*** - Handle to current or top design
- Input: `top` of type ***Boolean*** - Indicates if the call must return top (true) or current (false) designID

F.7.60.10 getGeneratorContextComponentInstanceID

Description: Returns the componentInstanceID associated with the currently invoked generator

- Returns: `componentInstanceID` of type ***String*** - Handle to componentInstance
- Input: `generatorName` of type ***String*** - Name of the currently invoked generator

F.7.60.11 getXML

Description: Returns XML fragment

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

— Returns: `xmlString` of type ***String*** - XML fragment
— Input: `elementID` of type ***String*** - IP-XACT XML element

F.7.60.12 message

Description: Send message level and message text to DE

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `severity` of type ***String*** - message level
— Input: `message` of type ***String*** - message text

F.7.60.13 init

Description: API initialization function Must be called before any other API call

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `apiVersion` of type ***String*** - Indicates the API version with which the generator is defined to work
— Input: `failureMode` of type ***String*** - Compatibility failure mode: fail—DE should return an error on the init call if its API version does not match the one passed to the init call; error—DE should return an error each time a potentially incompatible API call is made; warning—DE should increment a warning count each time a potentially incompatible API call is made
— Input: `message` of type ***String*** - Message that the DE may display to the user

F.7.60.14 end

Description: Terminate connection to the DE

— Returns: `status` of type ***Number*** - Status indicator from the DE Non-zero implies an error
— Input: `gen_status` of type ***Number*** - Status indicator from the generator Non-zero implies an error
— Input: `message` of type ***String*** - Message that the DE may display to the user

F.7.60.15 save

Description: Save all edits done in generator to DE; document must be valid

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

F.7.60.16 registerVLNV

Description: Register the VLNV contained in the given file, possibly replacing an existing VLNV

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `fileName` of type ***String*** - Location of file that is to be registered
— Input: `replace` of type ***Boolean*** - Replace existing VLNV with new content

F.7.60.17 unregisterVLNV

Description: Unregister the given VLNV

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `VLNV` of type ***String[]*** - VLNV that is to be unregistered

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.60.18 registerCatalogVLNVs

Description: Register all VLNVs in the given catalog

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `catalogID` of type ***String*** - Catalog who's VLNVs are to be registered

F.7.60.19 unregisterCatalogVLNVs

Description: Unregister all VLNVs in the given catalog

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `catalogID` of type ***String*** - Catalog who's VLNVs are to be unregistered

F.7.61 Indirect interface operations (BASE)

F.7.61.1 getIndirectInterfaceIndirectAddress

Description: Returns indirectAddress for the given indirectInterface element

- Returns: `indirectAddress` of type ***String*** - IndirectAddress value
- Input: `indirectInterfaceID` of type ***String*** - Handle to an indirectInterface element

F.7.61.2 getIndirectInterfaceIndirectData

Description: Returns indirectData for the given indirectInterface element

- Returns: `indirectData` of type ***String*** - IndirectData value
- Input: `indirectInterfaceID` of type ***String*** - Handle to an indirectInterface element

F.7.61.3 getIndirectInterfaceMemoryMapID

Description: Returns memoryMapID for the given indirectInterface element

- Returns: `memoryMapID` of type ***String*** - Handle to a memoryMap
- Input: `indirectInterfaceID` of type ***String*** - Handle to an indirectInterface element

F.7.61.4 getIndirectInterfaceBridgeIDs

Description: Returns bridgeIDs for the given indirectInterface element

- Returns: `bridgeIDs` of type ***String[]*** - List of bridge handles
- Input: `indirectInterfaceID` of type ***String*** - Handle to an indirectInterface element

F.7.61.5 getIndirectInterfaceBitSteering

Description: Returns indirectAddress for the given indirectInterface element

- Returns: `value` of type ***String*** - BitSteering value
- Input: `indirectInterfaceID` of type ***String*** - Handle to an indirectInterface element

F.7.61.6 getIndirectInterfaceEndianness

Description: Returns indirectAddress for the given indirectInterface element

- Returns: `value` of type ***String*** - Endianness value
- Input: `indirectInterfaceID` of type ***String*** - Handle to an indirectInterface element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.62 Indirect interface operations (EXTENDED)

F.7.62.1 addIndirectInterfaceBridge

Description: Add bridge with the given busInterfaceRef to the given indirectInterface element

- Returns: `bridgeID` of type ***String*** - Handle to a new bridge
- Input: `indirectInterfaceID` of type ***String*** - Handle to an indirectInterface element
- Input: `busInterfaceRef` of type ***String*** - Value for MasterRef attribute

F.7.62.2 removeIndirectInterfaceBridge

Description: Remove the given bridge element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `bridgeID` of type ***String*** - Handle to a bridge element

F.7.62.3 setIndirectInterfaceBitSteering

Description: Set the bitSteering with the given value for the given indirectInterface element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `indirectInterfaceID` of type ***String*** - Handle to an indirectInterface element
- Input: `value` of type ***String*** - BitSteering value

F.7.62.4 setIndirectInterfaceEndianness

Description: Set the endianness with the given value for the given indirectInterface element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `indirectInterfaceID` of type ***String*** - Handle to an indirectInterface element
- Input: `value` of type ***String*** - Endianness value

F.7.63 Memory map operations (BASE)

F.7.63.1 getMemoryMapElementIDs

Description: Returns the memoryMapElementIDs for the given memoryMap or localMemoryMap element

- Returns: `memoryMapElementIDs` of type ***String[]*** - List of memoryMapElement handles
- Input: `memoryMapID` | `localMemoryMapID` of type ***String*** - Handle to a memoryMap or localMemoryMap element

F.7.63.2 getMemoryMapElementType

Description: Returns the elementType for the given memoryMapElement element

- Returns: `memoryType` of type ***String*** - MemoryType (addressBlock, bank, subSpaceMap)
- Input: `memoryMapElementID` of type ***String*** - Handle to a memoryMapElement element

F.7.63.3 getAddressBlockBaseAddress

Description: Returns the baseAddress for the given addressBlock element

- Returns: `baseAddress` of type ***Number*** - AddressBlock base address
- Input: `addressBlockID` of type ***String*** - Handle to an addressBlock element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.63.4 getAddressBlockTypeIdentifier

Description: Returns the typeIdentifier for the given addressBlock element

- Returns: `typeIdentifier` of type ***String*** - AddressBlock type identifier
- Input: `addressBlockID` of type ***String*** - Handle to an addressBlock element

F.7.63.5 getAddressBlockRange

Description: Returns the range for the given address Block element

- Returns: `range` of type ***Number*** - AddressBlock range
- Input: `addressBlockID` of type ***String*** - Handle to an addressBlock element

F.7.63.6 getAddressBlockWidth

Description: Returns the width for the given address Block element

- Returns: `width` of type ***Number*** - AddressBlock width
- Input: `addressBlockID` of type ***String*** - Handle to an addressBlock element

F.7.63.7 getAddressBlockUsage

Description: Returns the usage for the given address Block element

- Returns: `usage` of type ***String*** - AddressBlock usage
- Input: `addressBlockID` of type ***String*** - Handle to an addressBlock element

F.7.63.8 getAddressBlockVolatility

Description: Returns the volatility for the given address Block element

- Returns: `volatile` of type ***Boolean*** - AddressBlock volatility
- Input: `addressBlockID` of type ***String*** - Handle to an addressBlock element

F.7.63.9 getAddressBlockAccess

Description: Returns the access for the given address Block element

- Returns: `access` of type ***String*** - AddressBlock access
- Input: `addressBlockID` of type ***String*** - Handle to an addressBlock element

F.7.63.10 getAddressBlockRegisterIDs

Description: Returns the registerIDs for the given address Block element

- Returns: `registerIDs` of type ***String[]*** - List of register handles
- Input: `addressBlockID` of type ***String*** - Handle to an addressBlock element

F.7.63.11 getAddressBlockRegisterFileIDs

Description: Returns the registerFileIDs for the given address Block element

- Returns: `registerFileIDs` of type ***String[]*** - List of registerFile handles
- Input: `addressBlockID` of type ***String*** - Handle to an addressBlock element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.63.12 getBankAlignment

Description: Returns the value of the bankAlignment attribute for the given bank or localBank element

- Returns: `alignment` of type ***String*** - Bank alignment
- Input: `bankID` | `localBankID` of type ***String*** - Handle to a bank or localBank element

F.7.63.13 getBankBaseAddress

Description: Returns the baseAddress for the given bank or localBank element

- Returns: `baseAddress` of type ***Number*** - Bank base address
- Input: `bankID` | `localBankID` of type ***String*** - Handle to a bank or localBank element

F.7.63.14 getBankBlockAndSubspaceElementIDs

Description: Returns the bankIDs, addressBlockIDs, and subSpaceMapIDs for the given bank or localBank element

- Returns: `memoryElementIDs` of type ***String[]*** - List of bank, addressBlock, and subspaceMap handles
- Input: `bankID` | `localBankID` of type ***String*** - Handle to a bank or localBank element

F.7.63.15 getBankUsage

Description: Returns the usage for the given bank or localBank element

- Returns: `usage` of type ***String*** - Bank usage
- Input: `bankID` | `localBankID` of type ***String*** - Handle to a bank or localBank element

F.7.63.16 getBankVolatility

Description: Returns the volatility for the given bank or localBank element

- Returns: `volatile` of type ***Boolean*** - Bank volatility
- Input: `bankID` | `localBankID` of type ***String*** - Handle to a bank or localBank element

F.7.63.17 getBankAccess

Description: Returns the access for the given bank or localBank element

- Returns: `access` of type ***String*** - Bank access
- Input: `bankID` | `localBankID` of type ***String*** - Handle to a bank or localBank element

F.7.63.18 getSubspaceMapMasterID

Description: Returns an associated busInterfaceID of the masterRef for the given subspaceMap element

- Returns: `busInterfaceID` of type ***String*** - Handle to a busInterface
- Input: `subspaceMapID` of type ***String*** - Handle to a subspaceMap element

F.7.63.19 getSubspaceMapSegmentID

Description: Returns an associated segmentID of the segmentRef for the given subspaceMap

- Returns: `segmentID` of type ***String*** - Handle to a segment
- Input: `subspaceMapID` of type ***String*** - Handle to a subspaceMap element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.63.20 getSubspaceMapBaseAddress

Description: Returns the baseAddress for the given subspaceMap

- Returns: `baseAddress` of type ***Number*** - SubspaceMap base address
- Input: `subspaceMapID` of type ***String*** - Handle to a subspaceMap element

F.7.63.21 getMemoryMapRemapIDs

Description: Returns the memoryRemapIDs for the given memoryMap or localMemoryMap element

- Returns: `memoryRemapIDs` of type ***String[]*** - List of memoryRemap handles
- Input: `memoryMapID` | `localMemoryMapID` of type ***String*** - Handle to a memoryMap or localMemoryMap element

F.7.63.22 getMemoryRemapState

Description: Returns the state for a given memoryRemap element

- Returns: `state` of type ***String*** - MemoryRemap state
- Input: `memoryRemapID` of type ***String*** - Handle to a memoryRemap element

F.7.63.23 getMemoryRemapElementIDs

Description: Returns the bankIDs, addressBlockIDs, and subSpaceMapIDs for the given memoryRemap element

- Returns: `memoryMapElementIDs` of type ***String[]*** - List of bank, addressBlock, and subspaceMap handles
- Input: `memoryRemapID` of type ***String*** - Handle to a memoryRemap element

F.7.63.24 getMemoryMapAddressUnitBits

Description: Returns the addressUnitBits for the given memoryMap or localMemoryMap element

- Returns: `addressUnitBits` of type ***Number*** - Memory map address unit bits
- Input: `memoryMapID` | `localMemoryMapID` of type ***String*** - Handle to a memoryMap or localMemoryMap element

F.7.63.25 getMemoryMapShared

Description: Returns the shared for the given memoryMap or localMemoryMap element

- Returns: `shared` of type ***String*** - Memory map shared
- Input: `memoryMapID` | `localMemoryMapID` of type ***String*** - Handle to a memoryMap or localMemoryMap element

F.7.64 Memory map operations (EXTENDED)

F.7.64.1 addMemoryMapAddressBlock

Description: Add an addressBlock with the given name, baseAddress, range, and width to the given memoryMap or localMemoryMap element

- Returns: `addressBlockID` of type ***String*** - Handle to a new addressBlock element
- Input: `memoryMapID` | `localMemoryMapID` of type ***String*** - Handle to a memoryMap or localMemoryMap element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

- Input: `name` of type ***String*** - AddressBlock name
- Input: `baseAddress` of type ***String*** - AddressBlock base address
- Input: `range` of type ***String*** - AddressBlock range
- Input: `width` of type ***String*** - AddressBlock width

F.7.64.2 removeMemoryMapAddressBlock

Description: Remove the given addressBlock element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `addressBlockID` of type ***String*** - Handle to an addressBlock element

F.7.64.3 setAddressBlockRange

Description: Set the range with the given value for the given addressBlock element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `addressBlockID` of type ***String*** - Handle to an addressBlock element
- Input: `range` of type ***String*** - AddressBlock range

F.7.64.4 setAddressBlockTypeIdentifier

Description: Set the typeIdentifier with the given value for the given addressBlock element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `addressBlockID` of type ***String*** - Handle to an addressBlock element
- Input: `typeIdentifier` of type ***String*** - AddressBlock typeIdentifier

F.7.64.5 setAddressBlockUsage

Description: Set the usage with the given value for the given addressBlock element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `addressBlockID` of type ***String*** - Handle to an addressBlock element
- Input: `usage` of type ***String*** - AddressBlock usage

F.7.64.6 setAddressBlockVolatility

Description: Set the volatility with the given value for the given addressBlock element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `addressBlockID` of type ***String*** - Handle to an addressBlock element
- Input: `volatile` of type ***Boolean*** - AddressBlock volatility

F.7.64.7 setAddressBlockAccess

Description: Set the access with the given value for the given addressBlock element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `addressBlockID` of type ***String*** - Handle to an addressBlock element
- Input: `access` of type ***String*** - AddressBlock access

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.64.8 addAddressBlockRegister

Description: Add a register with the given name, offset, and size, and a field with the given name, offset, and width to the given addressBlock

- Returns: `registerID` of type ***String*** - Handle to a new register element
- Input: `addressBlockID` of type ***String*** - Handle to an addressBlock element
- Input: `name` of type ***String*** - Register name
- Input: `offset` of type ***String*** - Register offset
- Input: `size` of type ***String*** - Register size
- Input: `fieldName` of type ***String*** - Field name
- Input: `fieldOffset` of type ***String*** - Field offset
- Input: `fieldWidth` of type ***String*** - Field width

F.7.64.9 removeAddressBlockRegister

Description: Remove the given register element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `registerID` of type ***String*** - Handle to a register element

F.7.64.10 addAddressBlockRegisterFile

Description: Add a registerFile with the given name, addressOffset, and range to the given addressBlock element

- Returns: `registerFileID` of type ***String*** - Handle to a new registerFile element
- Input: `addressBlockID` of type ***String*** - Handle to an addressBlock element
- Input: `name` of type ***String*** - RegisterFile name
- Input: `addressOffset` of type ***String*** - RegisterFile address offset
- Input: `range` of type ***String*** - RegisterFile range

F.7.64.11 removeAddressBlockRegisterFile

Description: Remove the given registerFile element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `registerFileID` of type ***String*** - Handle to a registerFile element

F.7.64.12 addMemoryMapBank

Description: Add a bank with the given name, baseAddress, and alignment to the given memoryMap or localMemoryMap element

- Returns: `bankID` of type ***String*** - Handle to a new bank element
- Input: `memoryMapID` | `localMemoryMapID` of type ***String*** - Handle to a memoryMap or localMemoryMap element
- Input: `name` of type ***String*** - Bank name
- Input: `baseAddress` of type ***String*** - Bank base address
- Input: `alignment` of type ***String*** - BankAlignment

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.64.13 removeMemoryMapBank

Description: Remove the given bank element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `bankID` | `localBankID` of type ***String*** - Handle to bank or localBank element

F.7.64.14 addBankBank

Description: Add a bank with the given name, baseAddress, and alignment to the given bank element

- Returns: `bankID` of type ***String*** - Handle to a new bank element
- Input: `bankID` | `localBankID` of type ***String*** - Handle to bank or localBank element
- Input: `name` of type ***String*** - Bank name
- Input: `baseAddress` of type ***String*** - Bank base address
- Input: `alignment` of type ***String*** - BankAlignment

F.7.64.15 removeBankBank

Description: Remove the given bank element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `bankID` | `localBankID` of type ***String*** - Handle to bank or localBank element

F.7.64.16 setBankUsage

Description: Set usage with the given value for the given bank element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `bankID` | `localBankID` of type ***String*** - Handle to bank or localBank element
- Input: `usage` of type ***String*** - Bank usage

F.7.64.17 setBankVolatility

Description: Set volatility with the given value for the given bank element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `bankID` | `localBankID` of type ***String*** - Handle to bank or localBank element
- Input: `volatile` of type ***Boolean*** - Bank volatility

F.7.64.18 setBankAccess

Description: Set access with the given value for the given bank element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `bankID` | `localBankID` of type ***String*** - Handle to bank or localBank element
- Input: `access` of type ***String*** - Bank access

F.7.64.19 addMemoryMapSubspaceMap

Description: Add a subspaceMap with the given name, masterRef, and baseAddress to the given memory map

- Returns: `subspaceMapID` of type ***String*** - Handle to a new subspaceMap element
- Input: `memoryMapID` of type ***String*** - Handle to a memoryMap element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

- Input: `name` of type ***String*** - SubspaceMap name
- Input: `masterRef` of type ***String*** - SubspaceMap masterRef
- Input: `baseAddress` of type ***String*** - SubspaceMap base address

F.7.64.20 removeMemoryMapSubspaceMap

Description: Remove the given subspaceMap element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `subspaceMapID` of type ***String*** - Handle to a subspaceMap element

F.7.64.21 setMemoryRemapSegmentRef

Description: Set the segmentRef with the given value for the given subspaceMap element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `subspaceMapID` of type ***String*** - Handle to a subspaceMap element
- Input: `segmentRef` of type ***String*** - SubspaceMap segmentRef

F.7.64.22 addMemoryMapRemapWithAddressBlock

Description: Add a memoryRemap with the given state and name containing an addressBlock with the given addressBlock name, baseAddress, range, and width to the given memoryMap or localMemoryMap element

- Returns: `memoryRemapID` of type ***String*** - Handle to a new memoryRemap element
- Input: `memoryMapID` | `localMemoryMapID` of type ***String*** - Handle to a memoryMap or localMemoryMap element
- Input: `state` of type ***String*** - MemoryRemap state
- Input: `remapName` of type ***String*** - MemoryRemap name
- Input: `name` of type ***String*** - AddressBlock name
- Input: `baseAddress` of type ***String*** - AddressBlock base address
- Input: `range` of type ***String*** - AddressBlock range
- Input: `width` of type ***Number*** - AddressBlock width

F.7.64.23 addMemoryMapRemapWithBank

Description: Add a memoryRemap with the given state and name containing a bank with the given bankName, baseAddress, and alignment to the given memoryMap or localMemoryMap element

- Returns: `memoryRemapID` of type ***String*** - Handle to a new memoryRemap element
- Input: `memoryMapID` | `localMemoryMapID` of type ***String*** - Handle to a memoryMap or localMemoryMap element
- Input: `state` of type ***String*** - MemoryRemap state
- Input: `remapName` of type ***String*** - MemoryRemap name
- Input: `name` of type ***String*** - Bank name
- Input: `baseAddress` of type ***String*** - Bank base address
- Input: `alignment` of type ***String*** - BankAlignment

F.7.64.24 addMemoryMapRemapWithSubspaceMap

Description: Add a memoryRemap with the given state and name containing a subspaceMap with the given subspaceName, baseAddress, and masterRef to the given memoryMap element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

- Returns: `memoryRemapID` of type ***String*** - Handle to a new memoryRemap element
- Input: `memoryMapID` of type ***String*** - Handle to a memoryMap element
- Input: `state` of type ***String*** - MemoryRemap state
- Input: `remapName` of type ***String*** - MemoryRemap name
- Input: `name` of type ***String*** - SubspaceMap name
- Input: `baseAddress` of type ***String*** - SubspaceMap base address
- Input: `masterRef` of type ***String*** - SubspaceMap masterRef

F.7.64.25 removeMemoryMapRemap

Description: Remove the given memoryRemap element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `memoryRemapID` of type ***String*** - Handle to a memoryRemap element

F.7.64.26 setMemoryMapShared

Description: Set the shared with the given value for the given memoryMap element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `memoryMapID` of type ***String*** - Handle to a memoryMap element
- Input: `shared` of type ***String*** - MemoryMap shared enumeration value

F.7.65 Module and type parameter operations (BASE)

F.7.65.1 getModuleOrTypeParameterIDs

Description: Returns moduleOrTypeParameterIDs of the given element

- Returns: `moduleOrTypeParameterIDs` of type ***String[]*** - List of moduleOrTypeParameter handles
- Input: `moduleOrTypeParameterContainerElementID` of type ***String*** - Handle to an element that has moduleOrTypeParameter elements

F.7.65.2 getModuleOrTypeParameterValue

Description: Returns value of the given moduleOrTypeParameter element

- Returns: `value` of type ***String*** - ModuleOrTypeParameter value
- Input: `moduleOrTypeParameterID` of type ***String*** - Handle to a moduleOrTypeParameter element

F.7.65.3 getModuleOrTypeParameterValueExpression

Description: Returns expression of the given moduleOrTypeParameter element

- Returns: `expression` of type ***String*** - ModuleOrTypeParameter expression
- Input: `moduleOrTypeParameterID` of type ***String*** - Handle to a moduleOrTypeParameter element

F.7.65.4 getModuleOrTypeParameterDataType

Description: Returns the data type value of the given moduleOrTypeParameter element

- Returns: `dataType` of type ***String*** - Datatype value

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

— Input: `moduleOrTypeParameterID` of type ***String*** - Handle to a moduleOrTypeParameter element

F.7.65.5 getModuleOrTypeParameterUsageType

Description: Returns the usage value of the given moduleOrTypeParameter element

— Returns: `usageType` of type ***String*** - Usage value

— Input: `moduleOrTypeParameterID` of type ***String*** - Handle to a moduleOrTypeParameter element

F.7.66 Module and type parameter operations (EXTENDED)

F.7.66.1 addModuleOrTypeParameter

Description: Add a moduleOrTypeParameter with the given name and given value to the given element

— Returns: `moduleOrTypeParameterID` of type ***String*** - Handle to a new moduleOrTypeParameter

— Input: `moduleOrTypeParameterContainerElementID` of type ***String*** - Handle to an element that has moduleOrTypeParameter elements

— Input: `name` of type ***String*** - moduleOrTypeParameter name

— Input: `expression` of type ***String*** - moduleOrTypeParameter expression

F.7.66.2 removeModuleOrTypeParameter

Description: Remove the given moduleOrTypeParameter

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

— Input: `moduleOrTypeParameterID` of type ***String*** - Handle to a moduleOrTypeParameter element

F.7.66.3 setModuleOrTypeParameterValue

Description: Set the value of the given moduleOrTypeParameter

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

— Input: `moduleOrTypeParameterID` of type ***String*** - Handle to a moduleOrTypeParameter element

— Input: `expression` of type ***String*** - moduleOrTypeParameter expression

F.7.66.4 setModuleOrTypeParameterDataType

Description: Set the data type value of the given moduleOrTypeParameter

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

— Input: `moduleOrTypeParameterID` of type ***String*** - Handle to a moduleOrTypeParameter element

— Input: `dataType` of type ***String*** - moduleOrTypeParameter data type value

F.7.66.5 setModuleOrTypeParameterUsageType

Description: Set the usage value of the given moduleOrTypeParameter

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

— Input: `moduleOrTypeParameterID` of type ***String*** - Handle to a moduleOrTypeParameter element
— Input: `usageType` of type ***String*** - moduleOrTypeParameter usage value

F.7.67 Parameter operations (BASE)

F.7.67.1 getParameterIDs

Description: Returns parameterIDs of the given element

— Returns: `parameterIDs` of type ***String[]*** - List of parameter handles
— Input: `parameterContainerElementID` of type ***String*** - Handle to an element that has parameter elements

F.7.67.2 getParameterValue

Description: Returns value of the given parameter element

— Returns: `value` of type ***String*** - Parameter value
— Input: `parameterID` of type ***String*** - Handle to a parameter element

F.7.67.3 getParameterValueExpression

Description: Returns expression of the given parameter element

— Returns: `expression` of type ***String*** - Parameter expression
— Input: `parameterID` of type ***String*** - Handle to a parameter element

F.7.67.4 addParameter

Description: Add a parameter with the given name and given value to the given element

— Returns: `parameterID` of type ***String*** - Handle to a new parameter
— Input: `parameterContainerElementID` of type ***String*** - Handle to an element that has parameter elements
— Input: `name` of type ***String*** - Parameter name
— Input: `expression` of type ***String*** - Parameter expression

F.7.67.5 removeParameter

Description: Remove the given parameter

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `parameterID` of type ***String*** - Handle to a parameter element

F.7.67.6 setParameterValue

Description: Set the value of the given parameter

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `parameterID` of type ***String*** - Handle to a parameter element
— Input: `expression` of type ***String*** - Parameter expression

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.68 Port operations (BASE)

F.7.68.1 getPortStyle

Description: Returns the style (wire or transactional) for the given port element

- Returns: `style` of type ***String*** - Port style (wire or transactional)
- Input: `portID` of type ***String*** - Handle to a port element

F.7.68.2 getPortAllLogicalDirectionsAllowed

Description: Returns allLogicalDirectionsAllowed for the given port element

- Returns: `value` of type ***Boolean*** - Port allLogicalDirectionsAllowed
- Input: `portID` of type ***String*** - Handle to a port element

F.7.68.3 getPortDirection

Description: Returns direction for the given port element

- Returns: `direction` of type ***String*** - Port direction
- Input: `portID` of type ***String*** - Handle to a port element

F.7.68.4 getPortWireTypeDefIDs

Description: Returns wireTypeDefIDs for the given port element

- Returns: `wireTypeDefIDs` of type ***String[]*** - List of wireTypeDef handles
- Input: `portID` of type ***String*** - Handle to a port element

F.7.68.5 getPortDriverIDs

Description: Returns driverIDs for the given port element

- Returns: `driverID` of type ***String[]*** - List of driver handles
- Input: `portID` of type ***String*** - Handle to a port element

F.7.68.6 getPortConstraintSetIDs

Description: Returns constraintSetIDs or the given port element

- Returns: `contraintSetIDs` of type ***String[]*** - List of constraintSet handles
- Input: `portID` of type ***String*** - Handle to a port element

F.7.68.7 getPortInitiative

Description: Returns initiative for the given port element

- Returns: `initiative` of type ***String*** - Port initiative
- Input: `portID` of type ***String*** - Handle to a port element

F.7.68.8 getPortKind

Description: Returns kind for the given port element

- Returns: `kind` of type ***String*** - Port kind
- Input: `portID` of type ***String*** - Handle to a port element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.68.9 getPortKindCustom

Description: Returns custom for the given port element

- Returns: `customKind` of type ***String*** - Port kind custom
- Input: `portID` of type ***String*** - Handle to a port element

F.7.68.10 getPortBusWidth

Description: Returns busWidth for the given port element

- Returns: `width` of type ***Number*** - Port busWidth
- Input: `portID` of type ***String*** - Handle to a port element

F.7.68.11 getPortProtocolType

Description: Returns protocolType for the given port element

- Returns: `type` of type ***String*** - Port protocolType
- Input: `portID` of type ***String*** - Handle to a port element

F.7.68.12 getPortProtocolTypeCustom

Description: Returns custom attribute of protocolType the given port element

- Returns: `customType` of type ***String*** - Port protocolType custom
- Input: `portID` of type ***String*** - Handle to a port element

F.7.68.13 getPortProtocolPayloadName

Description: Returns payloadName for the given port element

- Returns: `name` of type ***String*** - Port payloadName
- Input: `portID` of type ***String*** - Handle to a port element

F.7.68.14 getPortProtocolPayloadType

Description: Returns payloadType for the given port element

- Returns: `type` of type ***String*** - Port payloadType
- Input: `portID` of type ***String*** - Handle to a port element

F.7.68.15 getPortProtocolPayloadExtension

Description: Returns payloadExtension for the given port element

- Returns: `extension` of type ***String*** - Port payloadExtension
- Input: `portID` of type ***String*** - Handle to a port element

F.7.68.16 getPortProtocolPayloadExtensionMandatory

Description: Returns mandatory attribute of payloadExtension for the given port element

- Returns: `value` of type ***Boolean*** - Port payloadExtension mandatory
- Input: `portID` of type ***String*** - Handle to a port element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.68.17 getPortTransTypeDefIDs

Description: Returns transTypeDefIDs for the given port element

- Returns: `transTypeDefIDs` of type ***String[]*** - List of transTypeDef handles
- Input: `portID` of type ***String*** - Handle to a port element

F.7.68.18 getPortMaxConnections

Description: Returns maxConnections for the given port element

- Returns: `value` of type ***Number*** - Port max connections
- Input: `portID` of type ***String*** - Handle to a port element

F.7.68.19 getPortMinConnections

Description: Returns minConnections for the given port element

- Returns: `value` of type ***Number*** - Port min connections
- Input: `portID` of type ***String*** - Handle to a port element

F.7.68.20 getPortAccessType

Description: Returns accessType for the given port element

- Returns: `accessType` of type ***String*** - Port access type
- Input: `portID` of type ***String*** - Handle to a port element

F.7.69 Port operations (EXTENDED)

F.7.69.1 setPortAllLogicalDirectionsAllowed

Description: Set allLogicalDirectionsAllowed with the given value for the given port element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `portID` of type ***String*** - Handle to a port element
- Input: `value` of type ***Boolean*** - Port allLogicalDirectionAllowed value

F.7.69.2 addPortWireTypeDef

Description: Add wireTypeDef for the given port element

- Returns: `wireTypeDefID` of type ***String*** - Handle to a new wireTypeDef
- Input: `portID` of type ***String*** - Handle to a port element

F.7.69.3 removePortWireTypeDef

Description: Remove the given wireTypeDef element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `wireTypeDefID` of type ***String*** - Handle to a wireTypeDef element

F.7.69.4 addPortDefaultDriver

Description: Add defaultDriver with the given value to the given port element

- Returns: `driverID` of type ***String*** - Handle to a new driver

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

— Input: `portID` of type ***String*** - Handle to a port element
— Input: `value` of type ***Number*** - Default driver value

F.7.69.5 removePortDefaultDriver

Description: Remove the given defaultDriver element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `driverID` of type ***String*** - Handle to a driver element

F.7.69.6 addPortClockDriver

Description: Add clockDriver with the given clock period, pulse offset, pulse value, and pulse duration to the given port element

— Returns: `clockDriverID` of type ***String*** - Handle to a new clockDriver
— Input: `portID` of type ***String*** - Handle to a port element
— Input: `period` of type ***Number*** - Clock period
— Input: `offset` of type ***Number*** - Clock pulse offset
— Input: `value` of type ***Number*** - Clock pulse value
— Input: `duration` of type ***Number*** - Clock pulse duration

F.7.69.7 removePortClockDriver

Description: Remove the given clockDriver element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `clockDriverID` of type ***String*** - Handle to a clockDriver element

F.7.69.8 addPortSingleShotDriver

Description: Add singleShotDriver with the given offset, value, and duration to the given port element

— Returns: `singleShotDriverID` of type ***String*** - Handle to a new singleShotDriver
— Input: `portID` of type ***String*** - Handle to a port element
— Input: `offset` of type ***Number*** - SingleShot offset
— Input: `value` of type ***Number*** - SingleShot value
— Input: `duration` of type ***Number*** - SingleShot duration

F.7.69.9 removePortSingleShotDriver

Description: Remove the given singleShort driver element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `singleShotDriverID` of type ***String*** - Handle to a singleShot driver element

F.7.69.10 addPortConstraintSet

Description: Add constraintSet to the given port element

— Returns: `constraintSetID` of type ***String*** - Handle to a new constraintSet
— Input: `portID` of type ***String*** - Handle to a port element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.69.11 removePortConstraintSet

Description: Remove the given constraintSet element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `constraintSetID` of type ***String*** - Handle to a constraintSet element

F.7.69.12 setPortKind

Description: Set kind with the given value for the given port element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `portID` of type ***String*** - Handle to a port element
- Input: `kind` of type ***String*** - Port kind

F.7.69.13 setPortKindCustom

Description: Set attribute custom of kind with the given value for the given port element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `portID` of type ***String*** - Handle to a port element
- Input: `customKind` of type ***String*** - Port kind custom

F.7.69.14 setPortBusWidth

Description: Set busWidth with the given value for the given port element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `portID` of type ***String*** - Handle to a port element
- Input: `width` of type ***Number*** - Port busWidth

F.7.69.15 setPortProtocolType

Description: Set protocolType with the given value for the given port element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `portID` of type ***String*** - Handle to a port element
- Input: `type` of type ***String*** - Port type

F.7.69.16 setPortProtocolTypeCustom

Description: Set attribute custom of protocolType with the given value for the given port element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `portID` of type ***String*** - Handle to a port element
- Input: `customType` of type ***String*** - Port type custom

F.7.69.17 addPortProtocolPayload

Description: Add a payload with the given type to the given port element

- Returns: `protocolPayloadID` of type ***String*** - Handle to a new protocolPayloadID
- Input: `portID` of type ***String*** - Handle to a port element
- Input: `type` of type ***String*** - Payload type

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.69.18 removePortProtocolPayload

Description: Remove the given payload

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `protocolPayloadID` of type ***String*** - Handle to a protocolPayload element

F.7.69.19 setProtocolPayloadName

Description: Set payload name with the given value for the given port element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `protocolPayloadID` of type ***String*** - Handle to a protocolPayload element
- Input: `name` of type ***String*** - Payload name

F.7.69.20 setProtocolPayloadExtension

Description: Set payload extension with the given value for the given port element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `protocolPayloadID` of type ***String*** - Handle to a protocolPayload element
- Input: `extension` of type ***String*** - Payload extension

F.7.69.21 setProtocolPayloadExtensionMandatory

Description: Set attribute mandatory of payload extension with the given value for the given port element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `protocolPayloadID` of type ***String*** - Handle to a protocolPayload element
- Input: `mandatory` of type ***Boolean*** - Payload extension mandatory

F.7.69.22 addPortTransTypeDef

Description: Add transTypeDef for the given port element

- Returns: `transTypeDefID` of type ***String*** - Handle to a new transTypeDef
- Input: `portID` of type ***String*** - Handle to a port element

F.7.69.23 removePortTransTypeDef

Description: Remove the given transTypeDef element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `transTypeDefID` of type ***String*** - Handle to a transTypeDef element

F.7.69.24 setPortMaxConnections

Description: Set maxConnections with the given value for the given port element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `portID` of type ***String*** - Handle to a port element
- Input: `value` of type ***Number*** - Port maxConnections

F.7.69.25 setPortMinConnections

Description: Set minConnections with the given value for the given port element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `portID` of type ***String*** - Handle to a port element
- Input: `value` of type ***Number*** - Port minConnections

F.7.69.26 setPortAccessType

Description: Set portAccessType with the given value for the given port element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `portID` of type ***String*** - Handle to a port element
- Input: `accessType` of type ***String*** - Port PortAccessType

F.7.70 Present operations (BASE)

F.7.70.1 getIsPresentValue

Description: Returns the value of the isPresent element of the given element

- Returns: `value` of type ***Boolean*** - Value of isPresent element
- Input: `isPresentContainerElementID` of type ***String*** - Handle to an element that has an isPresent element

F.7.70.2 getIsPresentValueExpression

Description: Returns the expression of the isPresent element of the given element

- Returns: `expression` of type ***String*** - Expression of isPresent element
- Input: `isPresentContainerElementID` of type ***String*** - Handle to an element that has an isPresent element

F.7.70.3 setIsPresentValue

Description: Set the expression of the isPresent element of the given element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `isPresentContainerElementID` of type ***String*** - Handle to an element that has an isPresent element
- Input: `expression` of type ***String*** - New expression

F.7.70.4 addIsPresent

Description: Add an isPresent element to the given element with the given expression

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `isPresentContainerElementID` of type ***String*** - Handle to an element that has an isPresent element
- Input: `expression` of type ***String*** - New expression

F.7.70.5 removeIsPresent

Description: Remove the isPresent element from the given element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `isPresentContainerElementID` of type ***String*** - Handle to an element that has an isPresent element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.71 Register file operations (BASE)

F.7.71.1 getRegisterFileID

Description: Returns the value of the RegisterFileID attribute in the given registerFile element

- Returns: `id` of type ***String*** - RegisterFile id
- Input: `registerFileID` of type ***String*** - Handle to a registerFile element

F.7.71.2 getRegisterFileDimensions

Description: Returns the dim in the given registerFile element

- Returns: `dimensions` of type ***Number[]*** - RegisterFile dimensions
- Input: `registerFileID` of type ***String*** - Handle to a registerFile element

F.7.71.3 getRegisterFileAddressOffset

Description: Returns the addressOffset in the given registerFile element

- Returns: `addressOffset` of type ***Number*** - RegisterFile addressOffset
- Input: `registerFileID` of type ***String*** - Handle to a registerFile element

F.7.71.4 getRegisterFileTypeIdentifier

Description: Returns the typeIdentifier in the given registerFile element

- Returns: `typeIdentifier` of type ***String*** - RegisterFile typeIdentifier
- Input: `registerFileID` of type ***String*** - Handle to a registerFile element

F.7.71.5 getRegisterFileRange

Description: Returns the range in the given registerFile element

- Returns: `range` of type ***Number*** - RegisterFile range
- Input: `registerFileID` of type ***String*** - Handle to a registerFile element

F.7.72 Register file operations (EXTENDED)

F.7.72.1 setRegisterFileID

Description: Set the value of the RegisterFileID attribute in the given registerFile element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `registerFileID` of type ***String*** - Handle to a registerFile element
- Input: `id` of type ***String*** - Value of RegisterFileID attribute

F.7.72.2 setRegisterFileDimensions

Description: Set the dim with the given value n the given registerFile element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `registerFileID` of type ***String*** - Handle to a registerFile element
- Input: `dimensions` of type ***Number[]*** - RegisterFile dim

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.72.3 getRegisterFileTypeIdentifier

Description: Set the typeIdentifier with the given value n the given registerFile element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `registerFileID` of type ***String*** - Handle to a registerFile element
- Input: `typeIdentifier` of type ***String*** - RegisterFile typeidentifier

F.7.73 Register operations (BASE)

F.7.73.1 getRegisterDimensions

Description: Returns dim for the given register element

- Returns: `dimensions` of type ***Number[]*** - List of dimension values
- Input: `registerID` of type ***String*** - Handle to a register element

F.7.73.2 getRegisterAddressOffset

Description: Returns addressOffset for the given register element

- Returns: `addressOffset` of type ***Number*** - Register address offset
- Input: `registerID` of type ***String*** - Handle to a register element

F.7.73.3 getRegisterTypeIdentifier

Description: Returns typeIdentifier for the given register element

- Returns: `typeIdentifier` of type ***String*** - Register typeIdentifier
- Input: `registerID` of type ***String*** - Handle to a register element

F.7.73.4 getRegisterSize

Description: Returns size for the given register element

- Returns: `size` of type ***Number*** - Register size
- Input: `registerID` of type ***String*** - Handle to a register element

F.7.73.5 getRegisterVolatility

Description: Returns volatility for the given register element

- Returns: `volatile` of type ***Boolean*** - Register volatility
- Input: `registerID` of type ***String*** - Handle to a register element

F.7.73.6 getRegisterAccess

Description: Returns access for the given register element

- Returns: `access` of type ***String*** - Register access
- Input: `registerID` of type ***String*** - Handle to a register element

F.7.73.7 getRegisterFieldIDs

Description: Returns fieldIDs for the given register element

- Returns: `regFieldIDs` of type ***String[]*** - List of regField handles

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

— Input: `registerID` of type ***String*** - Handle to a register element

F.7.73.8 getRegisterAlternateRegisterIDs

Description: Returns alternativeRegisterIDs for the given register element

— Returns: `alternateRegisterIDs` of type ***String[]*** - List of alternateRegister handles

— Input: `registerID` of type ***String*** - Handle to a register element

F.7.73.9 getAlternateRegisterState

Description: Returns state for the given alternateRegister element

— Returns: `state` of type ***String*** - AlternateRegister state

— Input: `alternateRegisterID` of type ***String*** - Handle to a alternateRegister element

F.7.73.10 getAlternateRegisterGroups

Description: Returns groups for the given alternateRegister element

— Returns: `groups` of type ***String[]*** - AlternateRegister groups

— Input: `alternateRegisterID` of type ***String*** - Handle to a alternateRegister element

F.7.73.11 getAlternateRegisterTypeIdentifier

Description: Returns typeIdentifier for the given alternateRegister element

— Returns: `typeIdentifier` of type ***String*** - AlternateRegister typeIdentifier

— Input: `alternateRegisterID` of type ***String*** - Handle to a alternateRegister element

F.7.73.12 getAlternateRegisterVolatility

Description: Returns volatility for the given alternateRegister element

— Returns: `volatile` of type ***Boolean*** - AlternateRegister volatility

— Input: `alternateRegisterID` of type ***String*** - Handle to a alternateRegister element

F.7.73.13 getAlternateRegisterAccess

Description: Returns access for the given alternateRegister element

— Returns: `access` of type ***String*** - AlternateRegister access

— Input: `alternateRegisterID` of type ***String*** - Handle to a alternateRegister element

F.7.73.14 getAlternateRegisterFieldIDs

Description: Returns fieldIDs for the given alternateRegister element

— Returns: `regFieldIDs` of type ***String[]*** - List of regField handles

— Input: `alternateRegisterID` of type ***String*** - Handle to a alternateRegister element

F.7.73.15 getRegisterResetValue

Description: Returns resetValue derived from fieldResetValues for the given resetType and register element

— Returns: `value` of type ***Number*** - Register reset value

— Input: `registerID` of type ***String*** - Handle to a register element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

— Input: `resetTypeRef` of type ***String*** - ResetType name (null represents default type)

F.7.73.16 getRegisterResetMask

Description: Returns resetMask derived from fieldResetMasks for the given resetType and register element

— Returns: `mask` of type ***Number*** - Register reset mask
— Input: `registerID` of type ***String*** - Handle to a register element
— Input: `resetTypeRef` of type ***String*** - ResetType name (null represents default type)

F.7.73.17 getAlternateRegisterResetValue

Description: Returns resetValue derived from fieldResetValues for the given resetType and alternateRegister element

— Returns: `value` of type ***Number*** - AlternateRegister reset value
— Input: `alternateRegisterID` of type ***String*** - Handle to a alternateRegister element
— Input: `resetTypeRef` of type ***String*** - ResetType name (null represents default type)

F.7.73.18 getAlternateRegisterResetMask

Description: Returns resetMask derived from fieldResetMasks for the given resetType and alternateRegister element

— Returns: `mask` of type ***Number*** - AlternateRegister reset mask
— Input: `alternateRegisterID` of type ***String*** - Handle to a alternateRegister element
— Input: `resetTypeRef` of type ***String*** - ResetType name (null represents default type)

F.7.74 Register operations (EXTENDED)

F.7.74.1 setRegisterDimensions

Description: Set dim with the given value for the given register element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `registerID` of type ***String*** - Handle to a register element
— Input: `dimensions` of type ***Number[]*** - List of dimension values

F.7.74.2 setRegisterTypeIdentifier

Description: Set typeIdentifier with the given value for the given register element

— Returns: `status` of type Boolean - Indicates call is successful (true) or not (false)
— Input: `registerID` of type ***String*** - Handle to a register element
— Input: `typeIdentifier` of type ***String*** - Register typeIdentifier

F.7.74.3 setRegisterVolatility

Description: Set volatility with the given value for the given register element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `registerID` of type ***String*** - Handle to a register element
— Input: `volatile` of type ***Boolean*** - Register volatility

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.74.4 setRegisterAccess

Description: Set access with the given value for the given register element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `registerID` of type ***String*** - Handle to a register element
- Input: `access` of type ***String*** - Register access

F.7.74.5 addRegisterField

Description: Add regField with the given name, offset, and width to the given register element

- Returns: `regFieldID` of type ***String*** - Handle to a new regField element
- Input: `registerID` of type ***String*** - Handle to a register element
- Input: `name` of type ***String*** - RegField name
- Input: `offset` of type ***String*** - RegField offset
- Input: `width` of type ***String*** - RegField width

F.7.74.6 removeRegisterField

Description: Remove the given regField element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `regFieldID` of type ***String*** - Handle to a regField element

F.7.74.7 addRegisterAlternateRegister

Description: Add alternateRegister with the given groups to the given register element

- Returns: `alternateRegisterID` of type ***String*** - Handle to a new alternateRegister element
- Input: `registerID` of type ***String*** - Handle to a register element
- Input: `name` of type ***String*** - AlternateRegister name
- Input: `groups` of type ***String[]*** - AlternateRegister groups

F.7.74.8 removeRegisterAlternateRegister

Description: Remove the given alternateRegister element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `alternateRegisterID` of type ***String*** - Handle to a alternateRegister element

F.7.74.9 setAlternateRegisterState

Description: Set state with the given value for the given alternateRegister element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `alternateRegisterID` of type ***String*** - Handle to a alternateRegister element
- Input: `state` of type ***String*** - AlternateRegister state

F.7.74.10 setAlternateRegisterTypeIdentifier

Description: Set typeIdentifier with the given value for the given alternateRegister element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `alternateRegisterID` of type ***String*** - Handle to a alternateRegister element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

- Input: `typeIdentifier` of type ***String*** - AlternateRegister typeIdentifier

F.7.74.11 setAlternateRegisterVolatility

Description: Set volatility with the given value for the given alternateRegister element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `alternateRegisterID` of type ***String*** - Handle to a alternateRegister element
- Input: `volatile` of type ***Boolean*** - AlternateRegister volatility

F.7.74.12 setAlternateRegisterAccess

Description: Set access with the given value for the given alternateRegister element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `alternateRegisterID` of type ***String*** - Handle to a alternateRegister element
- Input: `access` of type ***String*** - AlternateRegister access

F.7.74.13 addAlternateRegisterField

Description: Add regField with the given name, offset, and width to the given alternateRegister element

- Returns: `regFieldID` of type ***String*** - Handle to a new regField element
- Input: `alternateRegisterID` of type ***String*** - Handle to a alternateRegister element
- Input: `name` of type ***String*** - RegField name
- Input: `offset` of type ***String*** - RegField offset
- Input: `width` of type ***String*** - RegField width

F.7.74.14 removeAlternateRegisterField

Description: Remove the given regField element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `regFieldID` of type ***String*** - Handle to a regField element

F.7.74.15 setRegisterResetValue

Description: Set the fieldResetValues with the given register reset value expression for the given resetType and register element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `registerID` of type ***String*** - Handle to a register element
- Input: `valueExpression` of type ***String*** - Register reset value expression
- Input: `resetTypeRef` of type ***String*** - ResetType name (null represents default type)

F.7.74.16 setRegisterResetMask

Description: Set the fieldResetMasks with the given register reset mask expression for the given resetType and register element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `registerID` of type ***String*** - Handle to a register element
- Input: `maskExpression` of type ***String*** - Register reset mask expression
- Input: `resetTypeRef` of type ***String*** - ResetType name (null represents default type)

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.74.17 setAlternateRegisterResetValue

Description: Set the fieldResetValues with the given alternateRegister reset value expression for the given resetType and alternateRegister element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `alternateRegisterID` of type ***String*** - Handle to a alternateRegister element
- Input: `valueExpression` of type ***String*** - AlternateRegister reset value expression
- Input: `resetTypeRef` of type ***String*** - ResetType name (null represents default type)

F.7.74.18 setAlternateRegisterResetMask

Description: Set the fieldResetMasks with the given alternateRegister reset mask expression for the given resetType and alternateRegister element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `alternateRegisterID` of type ***String*** - Handle to a alternateRegister element
- Input: `maskExpression` of type ***String*** - AlternateRegister reset mask expression
- Input: `resetTypeRef` of type ***String*** - ResetType name (null represents default type)

F.7.75 Remap state operations (EXTENDED)

F.7.75.1 getRemapStateRemapPortIDs

Description: Returns remapPortIDs for the given remapState element

- Returns: `remapPortIDs` of type ***String[]*** - List of remapPort handles
- Input: `remapStateID` of type ***String*** - Handle to a remapState element

F.7.75.2 getRemapPortPortID

Description: Returns the portID of the portRef for the given remapPort element

- Returns: `portID` of type ***String*** - Handle to a port element
- Input: `remapPortID` of type ***String*** - Handle to a remapPort element

F.7.75.3 getRemapPortPortIndex

Description: Returns the portIndex for the given remapPort element

- Returns: `portIndex` of type ***Number*** - remapPortID portIndex value
- Input: `remapPortID` of type ***String*** - Handle to a remapPort element

F.7.75.4 getRemapPortValue

Description: Returns the value for the given remapPort element

- Returns: `value` of type ***Number*** - remapPort value
- Input: `remapPortID` of type ***String*** - Handle to a remapPort element

F.7.75.5 addRemapStateRemapPort

Description: Add a remapPort with the given portRef and value to the given remapState element

- Returns: `remapPortID` of type ***String*** - Handle to a new remapPort element
- Input: `remapStateID` of type ***String*** - Handle to a remapState element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

- Input: `portRef` of type ***String*** - RemapPort PortRef value
- Input: `value` of type ***Number*** - RemapPort value

F.7.75.6 removeRemapStateRemapPort

Description: Remove the given remapPort element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `remapPortID` of type ***String*** - Handle to a remapPort element

F.7.75.7 setRemapPortPortIndex

Description: Set the portIndex with the given value for the given remapPort

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `remapPortID` of type ***String*** - Handle to a remapPort element
- Input: `portIndex` of type ***Number*** - RemapPort portIndex value

F.7.76 Slice operations (BASE)

F.7.76.1 getSlicePathSegmentIDs

Description: Returns the pathSegmentIDs for a given slice element

- Returns: `pathSegmentIDs` of type ***String[]*** - List of pathSegment handles
- Input: `sliceID` of type ***String*** - Handle to a slice element

F.7.76.2 getPathSegmentName

Description: Returns the pathSegmentName for a given pathSegment element

- Returns: `name` of type ***String*** - The value of the pathSegmentName
- Input: `pathSegmentID` of type ***String*** - Handle to a pathSegment element

F.7.76.3 getPathSegmentIndices

Description: Returns the indices for a given pathSegment element

- Returns: `indices` of type ***String[]*** - List of indices
- Input: `pathSegmentID` of type ***String*** - Handle to a pathSegment element

F.7.76.4 getSliceRange

Description: Returns the range for a given slice element

- Returns: `range` of type ***Number[]*** - Range with left at index 0 and right at index 1
- Input: `sliceID` of type ***String*** - Handle to a slice element

F.7.77 Slice operations (EXTENDED)

F.7.77.1 addSlicePathSegment

Description: Add a pathSegment with the given pathSegmentNames and indices to the given slice element

- Returns: `pathSegmentID` of type ***String*** - Handle to a new pathSegment
- Input: `sliceID` of type ***String*** - Handle to a slice element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

— Input: `pathSegmentName` of type ***String*** - path segment name
— Input: `indices` of type ***String[]*** - List of indices

F.7.77.2 removeSlicePathSegment

Description: Remove the given pathSegment element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `pathSegmentID` of type ***String*** - Handle to a pathSegment element

F.7.77.3 setSliceRange

Description: Set the range to the given range for the given slice element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `sliceID` of type ***String*** - Handle to a slice element
— Input: `range` of type ***String[]*** - Range with left at index 0 and right at index 1

F.7.78 Typedef operations (BASE)

F.7.78.1 getTypeDefTypeName

Description: Returns the typeName for the given typeDef element

— Returns: `typeName` of type ***String*** - TypeName value
— Input: `typeDefElementID` of type ***String*** - Handle to a typeDef element

F.7.78.2 getTypeDefTypeDefinitions

Description: Returns the typeDefinitions for the given typeDef element

— Returns: `typeDefinitions` of type ***String[]*** - List of typeDefinitions
— Input: `typeDefElementID` of type ***String*** - Handle to a typeDef element

F.7.78.3 getTypeDefTypeParameterIDs

Description: Returns the typeParameterIDs for the given transTypeDef or serviceTypeDef element

— Returns: `typeParameterIDs` of type ***String[]*** - List of typeParameter handles
— Input: `transTypeDefID` | `serviceTypeDefID` of type ***String*** - Handle to a transTypeDef or serviceTypeDef element

F.7.78.4 getTypeDefServiceTypeDefIDs

Description: Returns the serviceTypeDefIDs for the given transTypeDef element

— Returns: `serviceTypeDefIDs` of type ***String[]*** - List of serviceTypeDef handles
— Input: `transTypeDefID` of type ***String*** - Handle to a transTypeDef element

F.7.78.5 getTypeDefViewIDs

Description: Returns the viewIDs for the given typeDef element

— Returns: `viewIDs` of type ***String[]*** - List of view handles
— Input: `typeDefElementID` of type ***String*** - Handle to a typeDef element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.79 Typedef operations (EXTENDED)

F.7.79.1 setTypeDefTypeName

Description: Set the typeName with the given value for the given typeDef element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `typeDefElementID` of type ***String*** - Handle to a typeDef element
- Input: `typeName` of type ***String*** - Value of the typeName element

F.7.79.2 setTypeDefTypeDefinitions

Description: Set the typeDefinitions with the given definitions for the given typeDef element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `typeDefElementID` of type ***String*** - Handle to a typeDef element
- Input: `typeDefinition` of type ***String[]*** - List of type definitions

F.7.79.3 addTypeDefTypeParameter

Description: Add a typeParameter with the given name and value to the given typeDef element

- Returns: `typeParameterID` of type ***String*** - Handle to a new typeParameter
- Input: `transTypeDefID` | `serviceTypeDefID` of type ***String*** - Handle to a transTypeDef or serviceTypeDef element
- Input: `name` of type ***String*** - TypeParameter name
- Input: `expression` of type ***String*** - TypeParameter expression

F.7.79.4 removeTypeDefTypeParameter

Description: Remove the given typeParameter

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `typeParameterID` of type ***String*** - Handle to a typeParameter

F.7.79.5 addTypeDefServiceTypeDef

Description: Add a serviceTypeDef with the given typeName to the given typeDef element

- Returns: `serviceTypeDefID` of type ***String*** - Handle to a new serviceTypeDef
- Input: `transTypeDefID` of type ***String*** - Handle to a transTypeDef element
- Input: `typeName` of type ***String*** - ServiceDef typeName

F.7.79.6 removeTypeDefServiceTypeDef

Description: Remove the given serviceTypeDef

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `serviceTypeDefID` of type ***String*** - Handle to a typeParameter

F.7.79.7 addTypeDefViewName

Description: Add the given viewName to the viewRefs in the given typeDef element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `typeDefElementID` of type ***String*** - Handle to a typeDef element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

— Input: `viewName` of type String - Reference to a view

F.7.79.8 removeTypeDefViewName

Description: Removed the given viewName from the viewRefs in the given typeDef element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

— Input: `typeDefElementID` of type ***String*** - Handle to a typeDef element

— Input: `viewName` of type ***String*** - Reference to a view

F.7.80 Vector operations (BASE)

F.7.80.1 getVectorIDs

Description: Returns vectorIDs of the given element

— Returns: `vectorIDs` of type ***String[]*** - List of vector handles

— Input: `vectorContainerElementID` of type ***String*** - Handle to an element that has a vector element

F.7.80.2 getVectorRange

Description: Returns range of the given vector element

— Returns: `range` of type ***Number[]*** - Range with left in index 0 and right in index 1

— Input: `vectorID` of type ***String*** - Handle to a vector element

F.7.80.3 getVectorRangeExpression

Description: Returns the range expressions of the given vector element

— Returns: `rangeExpression` of type ***String[]*** - Range expression with left index 0 and right in index 1

— Input: `vectorID` of type ***String*** - Handle to a vector element

F.7.81 Vector operations (EXTENDED)

F.7.81.1 addVector

Description: Add a vector to the given element

— Returns: `vectorID` of type String - Handle to new vector

— Input: `vectorContainerElementID` of type ***String*** - Handle to an element that has a vector element

— Input: `range` of type ***String[]*** - Range expression with left in index 0 and right in index 1

F.7.81.2 removeVector

Description: Remove the given vector

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

— Input: `vectorID` of type ***String*** - Handle to a vector element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.82 Vendor extensions operations (BASE)

F.7.82.1 getVendorExtensions

Description: Returns vendorExtensionID of the given element

- Returns: `vendorExtensions` of type ***String*** - VendorExtension handle
- Input: `vendorExtensionsContainerElementID` of type ***String*** - Handle to an element that has a vendor extension element

F.7.82.2 addVendorExtensions

Description: Add a vendor extension with given value to the given element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `vendorExtensionsContainerElementID` of type ***String*** - Handle to an element that has a vendor extension element
- Input: `vendorExtensions` of type ***String*** - Vendor extension value

F.7.82.3 removeVendorExtensions

Description: Remove the given vendor extension

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `vendorExtensionsContainerElementID` of type ***String*** - Handle to an element that has a vendor extension element

F.7.82.4 setVendorExtensions

Description: Set the value of the given vendor extension

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `vendorExtensionsContainerElementID` of type ***String*** - Handle to an element that has a vendor extension element
- Input: `vendorExtensions` of type ***String*** - Vendor extension value

F.7.83 View operations (BASE)

F.7.83.1 getViewEnvIdentifier

Description: Returns envIdentifier for the given view element

- Returns: `envIdentifier` of type ***String[]*** - View envIdentifier
- Input: `viewID` of type ***String*** - Handle to a view element

F.7.83.2 getViewComponentInstantiationRef

Description: Returns componentInstantiationRef for the given view element

- Returns: `componentInstantiationName` of type ***String*** - Name of componentInstantiation
- Input: `viewID` of type ***String*** - Handle to a view element

F.7.83.3 getViewComponentInstantiationID

Description: Returns componentInstantiationID for the given view element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

— Returns: `componentInstantiationID` of type ***String*** - Handle to a componentInstantiation
— Input: `viewID` of type ***String*** - Handle to a view element

F.7.83.4 getViewDesignInstantiationRef

Description: Returns designInstantiationRef for the given view element
— Returns: `designInstantiationName` of type ***String*** - Name of designInstantiation
— Input: `viewID` of type ***String*** - Handle to a view element

F.7.83.5 getViewDesignInstantiationID

Description: Returns designInstantiationID for the given view element
— Returns: `designInstantiationID` of type ***String*** - Handle to a designInstantiation
— Input: `viewID` of type ***String*** - Handle to a view element

F.7.83.6 getViewDesignConfigurationInstantiationRef

Description: Returns designConfigurationInstantiationRef for the given view element
— Returns: `designConfigurationInstantiationName` of type ***String*** - Name of designConfigurationInstantiation
— Input: `viewID` of type ***String*** - Handle to a view element

F.7.83.7 getViewDesignConfigurationInstantiationID

Description: Returns designConfigurationInstantiationID for the given view element
— Returns: `designConfigurationInstantiationID` of type ***String*** - Handle to a designConfiguredInstantiation
— Input: `viewID` of type ***String*** - Handle to a view element

F.7.84 View operations (EXTENDED)

F.7.84.1 setViewComponentInstantiationRef

Description: Set componentInstantiationRef for the given view element
— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `viewID` of type ***String*** - Handle to a view element
— Input: `componentInstantiationRef` of type ***String*** - ComponentInstantiation name

F.7.84.2 setViewDesignInstantiationRef

Description: Set designInstantiationRef for the given view element
— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
— Input: `viewID` of type ***String*** - Handle to a view element
— Input: `designInstantiationRef` of type ***String*** - DesignInstantiation name

F.7.84.3 setViewDesignConfigurationInstantiationRef

Description: Set designConfigurationInstantiationRef for the given view element
— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

- Input: `viewID` of type ***String*** - Handle to a view element
- Input: `designConfigurationInstantiationRef` of type ***String*** - DesignConfigurationInstantiation name

F.7.85 Whitebox operations (BASE)

F.7.85.1 getWhiteboxElementType

Description: Returns whiteboxType for the given whiteboxElement

- Returns: `type` of type ***String*** - Whitebox whiteboxType
- Input: `whiteboxElementID` of type ***String*** - Handle to a whiteboxElement element

F.7.85.2 getWhiteboxElementDrivable

Description: Returns driveable for the given whiteboxElement

- Returns: `value` of type ***Boolean*** - Whitebox driveable
- Input: `whiteboxElementID` of type ***String*** - Handle to a whiteboxElement

F.7.86 Whitebox operations (EXTENDED)

F.7.86.1 setWhiteboxElementDrivable

Description: Set driveable with the given value for the given whiteBox element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `whiteboxElementID` of type ***String*** - Handle to a whiteboxElement element
- Input: `value` of type ***Boolean*** - WhiteboxElement driveable

F.7.87 Whitebox ref operations (BASE)

F.7.87.1 getWhiteboxElementRefID

Description: Returns the value of the attribute name in the given whiteboxElementRef element

- Returns: `whiteboxElementID` of type ***String*** - Value of the attribute name
- Input: `whiteboxElementRefID` of type ***String*** - Handle to a whiteboxElementRef element

F.7.87.2 getWhiteboxElementRefLocationIDs

Description: Returns the whiteboxElementReflocationIDs for the given whiteboxElementRef element

- Returns: `whiteboxElementRefLocationIDs` of type ***String[]*** - List of whiteboxElementRefLocation handles
- Input: `whiteboxElementRefID` of type ***String*** - Handle to a whiteboxElementRef element

F.7.87.3 getWhiteboxElementRefLocationSlicesIDs

Description: Returns the slideIDs for the given whiteboxElementRefLocationID

- Returns: `sliceIDs` of type ***String[]*** - List of slices
- Input: `whiteboxElementRefLocationID` of type ***String*** - Handle to a whiteboxElementRefLocation element

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.88 Whitebox ref operations (EXTENDED)

F.7.88.1 addWhiteboxElementRefLocation

Description: Adds a whiteboxElementRefLocation element to a given whiteboxElementRef element

- Returns: `whiteboxElementRefLocationID` of type ***String*** - Handle to a new whiteboxElementRefLocation
- Input: `whiteboxElementRefID` of type ***String*** - Handle to a whiteboxElementRef element
- Input: `pathSegmentName` of type ***String*** - path segment name
- Input: `indices` of type ***String[]*** - List of indices

F.7.88.2 removeWhiteboxElementRefLocation

Description: Removes the given whiteboxElementRefLocation element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `whiteboxElementRefLocationID` of type ***String*** - Handle to a whiteboxElementRefLocation element

F.7.88.3 addWhiteboxElementRefLocationSlice

Description: Adds a slice to a given whiteboxElementRefLocation element

- Returns: `sliceID` of type ***String*** - Handle to a new slice
- Input: `whiteboxElementRefLocationID` of type ***String*** - Handle to a whiteboxElementRefLocation element
- Input: `pathSegmentName` of type ***String*** - path segment name
- Input: `indices` of type ***String[]*** - List of indices

F.7.88.4 removeWhiteboxElementRefLocationSlice

Description: Removes the given slice element

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `sliceID` of type ***String*** - Handle to a slice element

F.7.89 name group operations (BASE)

F.7.89.1 getName

Description: Returns name of the given element

- Returns: `name` of type ***String*** - Name of the given element
- Input: `nameGroupContainerElementID` of type ***String*** - Handle to an element that has a nameGroup

F.7.89.2 getDisplayName

Description: Returns display name of the given element

- Returns: `displayName` of type ***String*** - Display name of the given element
- Input: `nameGroupContainerElementID` | `generatorChainID` of type ***String*** - Handle to an element that has a nameGroup

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.89.3 getDescription

Description: Returns description of the given element

- Returns: `description` of type ***String*** - Description of the given element
- Input: `nameGroupContainerElementID` | topElementID of type ***String*** - Handle to an element that has a nameGroup

F.7.89.4 addDisplayName

Description: Add given display name to the given element; fails if display name is already set

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `nameGroupContainerElementID` | `generatorChainID` of type ***String*** - Handle to an element that has a nameGroup
- Input: displayName of type ***String*** - New display name

F.7.89.5 removeDisplayName

Description: Remove display name from the given element; fails if display name is not already set

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `nameGroupContainerElementID` | `generatorChainID` of type ***String*** - Handle to an element that has a nameGroup

F.7.89.6 setDisplayName

Description: Set given display name for the given element; fails if display name is not already set

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `nameGroupContainerElementID` | `generatorChainID` of type ***String*** - Handle to an element that has a nameGroup
- Input: `displayName` of type ***String*** - New display name

F.7.89.7 addDescription

Description: Add given description to the given element; fails if description is already set

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `nameGroupContainerElementID` | `topElementID` of type ***String*** - Handle to an element that has a nameGroup
- Input: `description` of type ***String*** - New description

F.7.89.8 removeDescription

Description: Remove description from the given element; fails if description is not already set

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)
- Input: `nameGroupContainerElementID` | `topElementID` of type ***String*** - Handle to an element that has a nameGroup

F.7.89.9 setDescription

Description: Set given description for the given element; fails if description is not already set

- Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

— Input: `nameGroupContainerElementID` | `topElementID` of type ***String*** - Handle to an element that has a nameGroup

— Input: `description` of type ***String*** - New description

F.7.90 name group operations (EXTENDED)

F.7.90.1 setName

Description: Set name of the given element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

— Input: `nameGroupContainerElementID` of type ***String*** - Handle to an element that has a nameGroup

— Input: `name` of type ***String*** - New name

F.7.91 top element operations (BASE)

F.7.91.1 getID

Description: Returns ID of the element with the given VLNV or null if this element does not exist

— Returns: `topElementID` of type ***String*** - Handle to element with given VLNV

— Input: `VLNV` of type ***String[]*** - List of string values representing a VLNV

F.7.91.2 getVLNV

Description: Returns VLNV of the element with the given ID

— Returns: `VLNV` of type ***String[]*** - VLNV of the element with the given ID

— Input: `topElementID` of type ***String*** - Handle to a top-level element

F.7.91.3 getXMLPath

Description: Returns location of the XML file that is registered with the VLNV of the given top-element

— Returns: `path` of type String - Location of the XML file

— Input: `topElementID` of type ***String*** - Handle to a top-level element

— Input: `dereference` of type ***Boolean*** - Indicates if links are to be dereferenced

F.7.91.4 edit

Description: Signal start of editing of the given top-element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

— Input: `topElementID` of type ***String*** - Handle to a top-level element

F.7.91.5 setXMLPath

Description: Set new location for the XML file that is registered with the VLNV of the given top-element

— Returns: `status` of type ***Boolean*** - Indicates call is successful (true) or not (false)

— Input: `topElementID` of type ***String*** - Handle to a top-level element

— Input: `path` of type ***String*** - New location of XML file

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

F.7.92 All ID types

This subclause lists the TGI ID types.

abstractionDefID	**abstractionDefInstanceID**	**abstractionDefPortID**	**abstractionDefPortModeConstraintID**
abstractionDefPortModeID	**abstractionTypeID**	**abstractorBusInterfaceID**	**abstractorID**
abstractorInstanceID	**abstractorInstancesID**	**abstractorInstancesInterfaceID**	**abstractorViewID**
activeInterfaceID	**adHocConnectionID**	**adHocExernalPortReferenceID**	**adHocExternalPortReferenceID**
adHocInternalPortReferenceID	**addressBlockID**	**addressSpaceID**	**alternateRegisterID**
argumentID	**arraryID**	**arrayID**	**assertionID**
bankID	**bridgeID**	**busDefID**	**busDefInstanceID**
busInterfaceID	**catalogID**	**channelID**	**choiceEnumerationID**
choiceID	**clockDriverID**	**componentID**	**componentInstanceID**
componentInstantiationID	**configurableElementID**	**configurableElementValueID**	**constraintSetID**
constraintSetRefID	**contraintSetID**	**contraintSetRefID**	**cpuID**
designConfigurationID	**designConfigurationInstantiationID**	**designID**	**designInstanceID**
designInstantiationID	**driveConstraintID**	**driveContraintID**	**driverID**
enumeratedValueID	**executableImageID**	**fileBuilderID**	**fileDefineID**
fileID	**fileSetGroupID**	**fileSetID**	**fileSetRefID**
functionID	**functionSourceFileID**	**generatorChainConfigurationID**	**generatorChainID**
generatorID	**groupSelectorID**	**hierInterfaceID**	**indirectInterfaceID**
interconnectionConfigurationID	**interconnectionID**	**ipxactFileID**	**linkerCommandFileID**
loadConstraintID	**localAddressBlockID**	**localBankID**	**localMemoryMapID**
logicalPortID	**memoryElementID**	**memoryMapElementID**	**memoryMapID**
memoryRemapID	**moduleOrTypeParameterID**	**monitorInterconnectionID**	**monitorInterfaceID**
monitoredInterfaceID	**parameterID**	**pathSegmentID**	**physicalPortID**
portID	**portMapID**	**protocolPayloadID**	**referenceID**
regFieldID	**registerFileID**	**registerID**	**remapAddressID**
remapPortID	**remapStateID**	**resetID**	**resetTypeID**
segmentID	**serviceTypeDefID**	**singleShotDriverID**	**sliceID**
subspaceMapID	**timingConstraintID**	**timingContraintID**	**topElementID**
transTypeDefID	**typeParameterID**	**unconfiguredElementID**	**vectorID**
viewConfigurationID	**viewID**	**whiteboxElementID**	**whiteboxElementRefID**
whiteboxElementRefLocationID	**wireTypeDefID**		

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Annex G

(informative)

External bus with an internal/digital interface

While the current use of IP-XACT schema may be viewed as describing single chip implementations, the schemas works equally well at the package- and board-level. Often a PHY component exists that interconnects the internal and external bus. Some interface standards define both of these interfaces, some define only the internal, and some define only the external. A common point of confusion is to use an external bus standard as an interface on an internal component. This is legal if the component caries the full PHY implementation, but this often makes the component very technology- or implementation-dependant.

G.1 Example: ethernet interfaces

An Ethernet bus might be described as more than a single wire, and in a system that includes Ethernet buses, it might also include all the interfaces shown in Figure G.1.

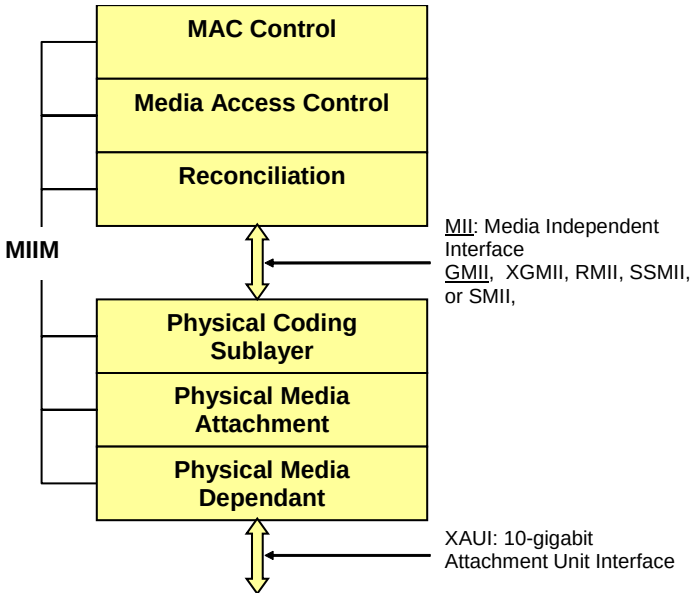


Figure G.1—Ethernet interface examples

XAUI: 10-gigabit Attachment Unit Interface

MII: Media Independent Interface

GMII: Gigabit Media Independent Interface

XGMII: 10-gigabit media-independent interface

RMII: Reduced MII, 7-pin interface

SSMII: Source Synchronous MII

SMII: Serial Media Independent Interface, this provides an interface to Ethernet MAC. The SMII provides the same interface as the MII, but with a reduced pin-out. The reduction in ports is

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

achieved by multiplexing data and control information to a port transmit port and a single receive port.

G.2 Example: I²C bus

The I²C bus is a two-wire bus with a clock and data line. The standard described bus is the two-wire bus. IP-XACT has defined an additional, related bus that is the internal digital interface. The internal digital interface shown in Figure G.2 contains three pins for each external pin: for `SDA` (the data line), the internal pins are defined as input, output, and enable as `SDA_I`, `SDA_O`, and `SDA_E`; in a similar manner, for the clock bus `SCL`, the internal pins are defined again for the functions of input, output, and enable as `SCL_I`, `SCL_O`, and `SCL_E`.

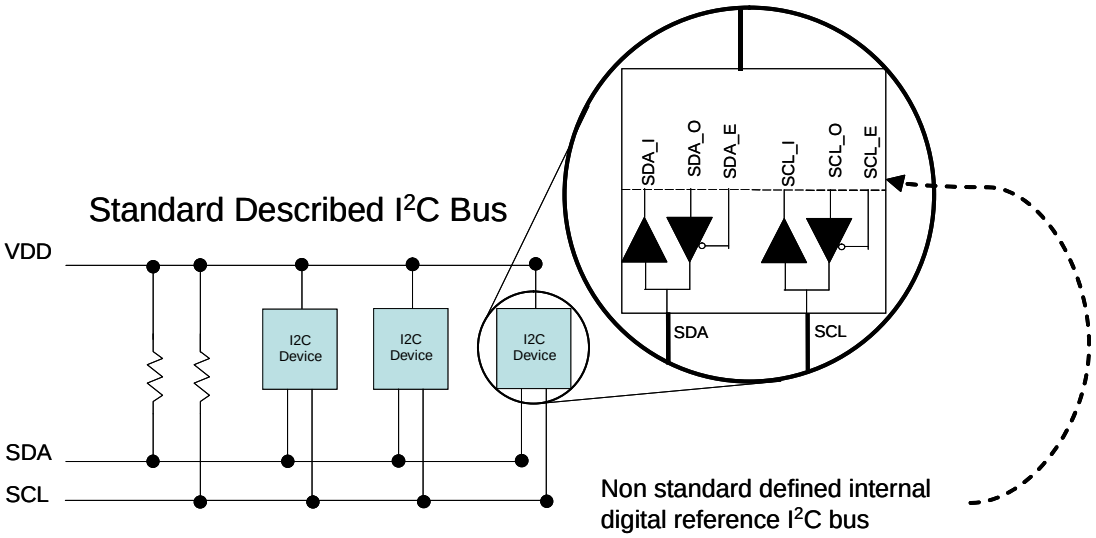


Figure G.2—I²C interface example

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Annex H

(informative)

Bridges and channels

This annex describes the basic address calculations of the two interconnect schemes contained inside an IP-XACT component: a **bridge** statement that describes an interconnect between a slave interface and a master interface, and a **channel** statement that describes an interconnect between a mirrored-master interface and a mirrored-slave interface. Figure H.1 highlights bridge and channel components in IP-XACT. For precise details on the addressing equations, see Clause 12.

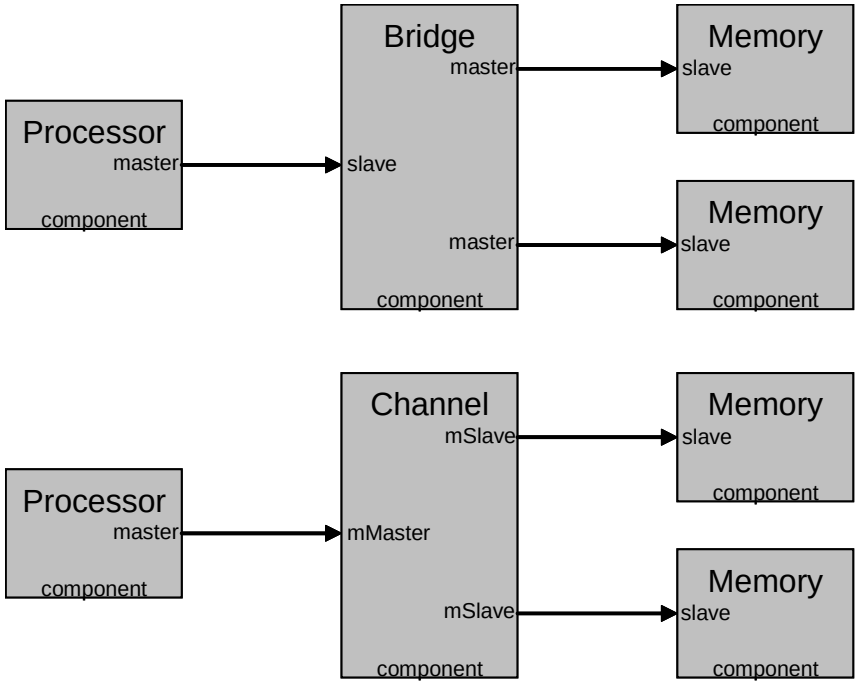


Figure H.1—Bridge and channel components

H.1 Transparent bridge

A transparent bridge locates the start of the master interface's address space at the start of the address space seen at the slave interface; thus, the address is not modified from the slave interface of the bridge into the **addressSpace** of the master interface. In Figure H.2, the master interface address space range `0x0000` to `0x0FFF` maps to the address range `0x0000` to `0x0FFF` as seen in the address space at the slave interface.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

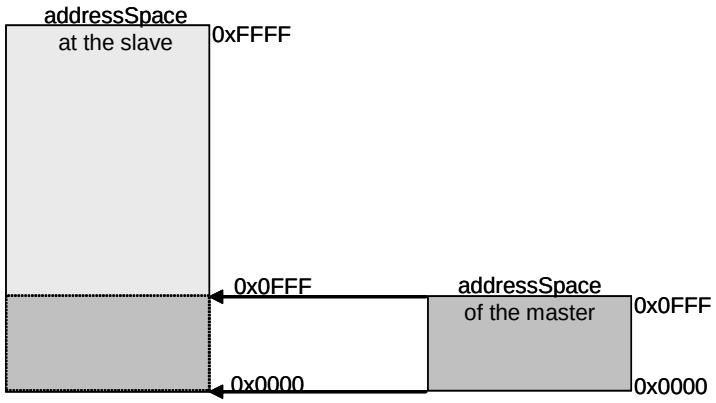


Figure H.2—Transparent bridge slave interface address range

An address block from another component connected to the bridge's master interface may appear in the master's address space. The base address of the connected address block is offset in the address space of the master interface by the **master**/**addressSpaceRef**/**baseAddress**. This also offsets the address block by the same amount in the address space at the slave interface. In Figure H.3, the **addressBlock** from the connected slave range `0x0000` to `0x07FF` maps to the address range `0x0600` to `0x0DFF` (offset by **master**/**addressSpaceRef**/**baseAddress** = `0x0600`) as seen in the address space of the master interface and to the address range `0x0600` to `0x0DFF` as seen in the address space at the slave interface.

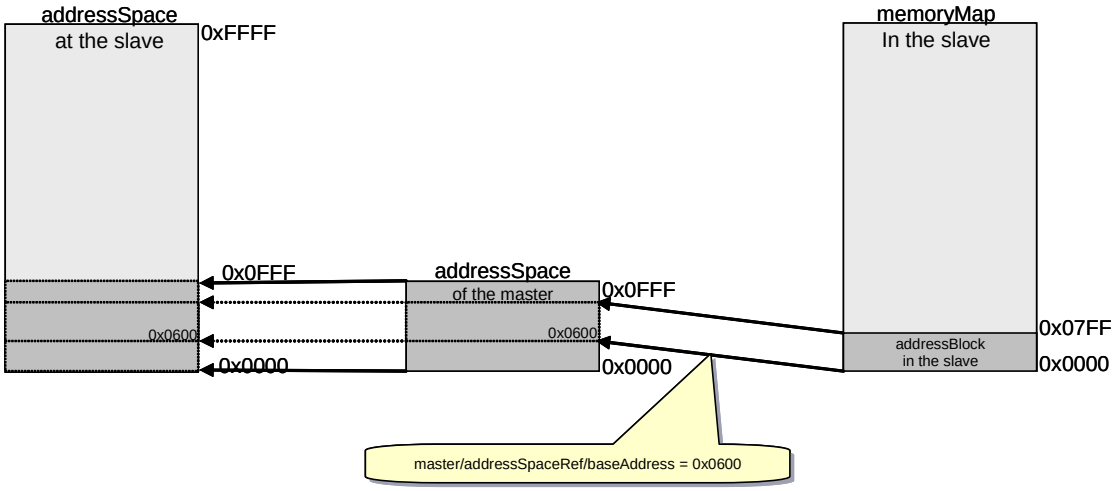


Figure H.3—Offsetting an address block in a transparent bridge

Figure H.4 shows it is also possible to offset the **addressBlock** from the connected slave in the negative direction. The **addressBlock** from the connected slave range `0x7000` to `0x77FF` maps to the address range `0x0000` to `0x07FF` (offset by **master**/**addressSpaceRef**/**baseAddress** = `–0x7000`) as seen in the address space of the master interface and to the address range `0x0000` to `0x07FF` as seen in the address space at the slave interface.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

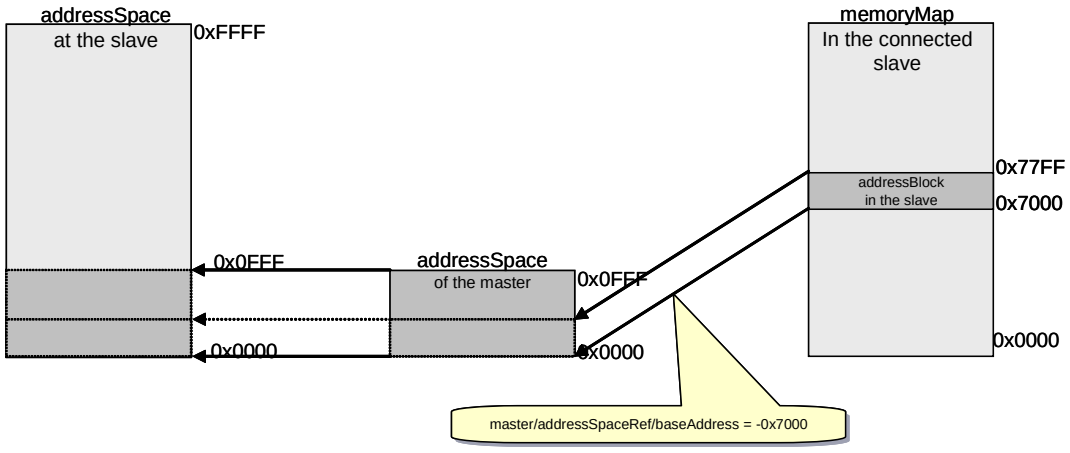


Figure H.4—Negative offsetting of an address block in a transparent bridge

Figure H.5 shows the references between the various elements and attributes in a transparent bridge.

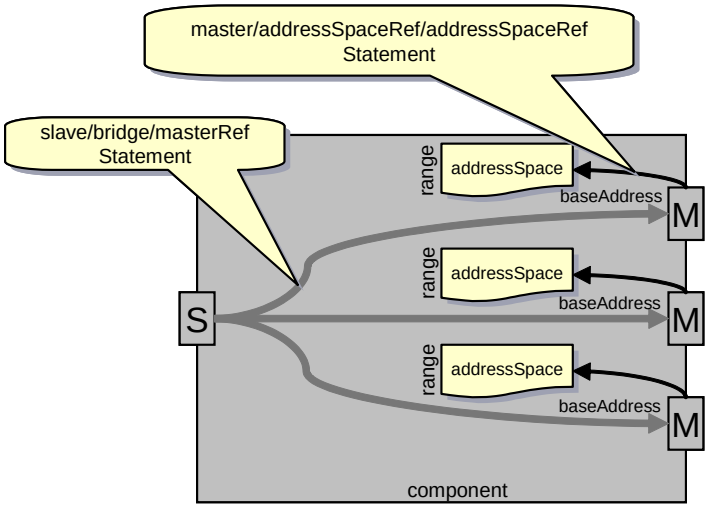


Figure H.5—Transparent bridge references

H.2 Opaque bridge

H.2.1 Without an address space segment reference

An opaque bridge that only references a master interface locates the start of the master interface's address space at the base address specified in the subspace map referenced by the slave interface; thus, the address is modified (offset by **subspaceMap**/**baseAddress**) from the slave interface into the **addressSpace** of the master interface. In Figure H.6, the slave interface address range `0x1000` to `0x1FFF` maps to address range `0x0000` to `0x0FFF` in the master interface's address space. The **range** of the addresses mapped is determined by the **range** of the **addressSpace**.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

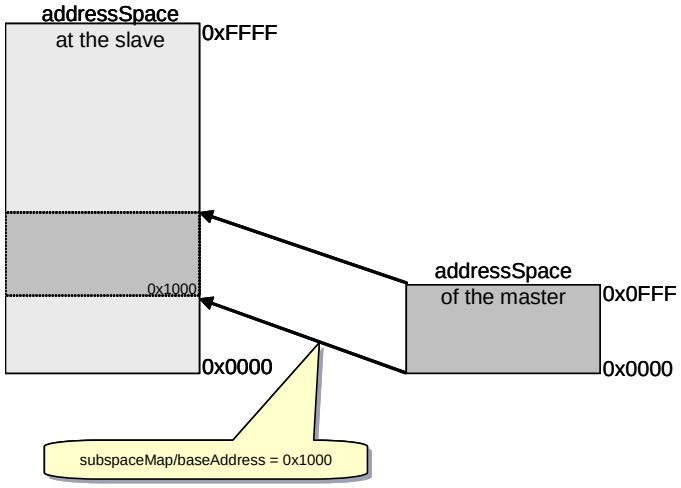


Figure H.6—Opaque bridge slave interface address range without segment reference

H.2.2 With an address space segment reference

An opaque bridge with an **addressSpace** segment reference locates the start of the master interface's address space segment at the base address specified in the subspace map referenced by the slave interface; thus, the address is modified (offset by **subspaceMap**/**baseAddress**) from the slave interface into the **addressSpace** of the master interface. In Figure H.7, the slave interface address range `0x1000` to `0x17FF` maps to address range `0x2000` to `0x27FF` in the master interface's address space. The **range** of the addresses mapped is determined by the **range** of the address space's segment.

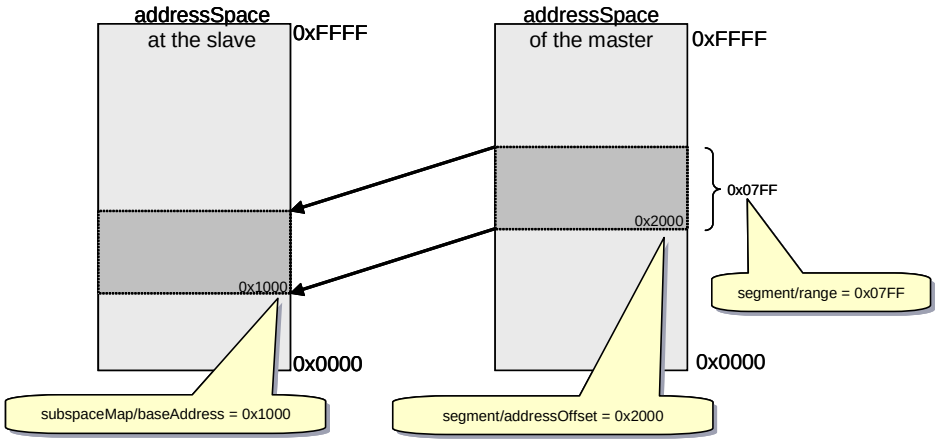


Figure H.7—Opaque bridge slave interface address range with segment reference

It is also possible to preserve the addressing across an opaque bridge. In Figure H.8, the slave interface address range `0x1000` to `0x17FF` maps to address range `0x1000` to `0x17FF` in the master interface's address space. The **range** of the addresses mapped is determined by the **range** of the address space's segment.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

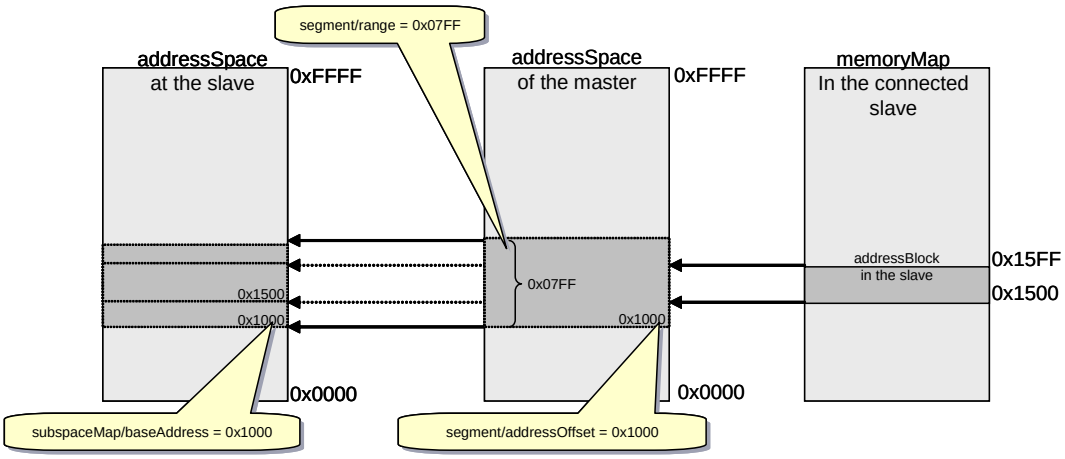


Figure H.8—Opaque bridge with transparent addressing

H.2.3 Effect of a master interface address space base address

The effect of the master interface address space base address applies with or without a segment reference. An address block from another component connected to the bridge's master interface may appear in the master's address space. The base address of the connected address block is offset in the address space of the master interface by the **master**/**addressSpaceRef**/**baseAddress**. This also offsets the address block by the same amount in the address space at the slave interface. In Figure H.9, the **addressBlock** from the connected slave range `0x0000` to `0x07FF` maps to the address range `0x0500` to `0x0CFF` (offset by **master**/**addressSpaceRef**/**baseAddress** = `0x0500`) as seen in the address space of the master interface, and to the address range `0x1500` to `0x1CFF` (offset by **subspaceMap**/**baseAddress** = `0x1000`) as seen in the address space at the slave interface.

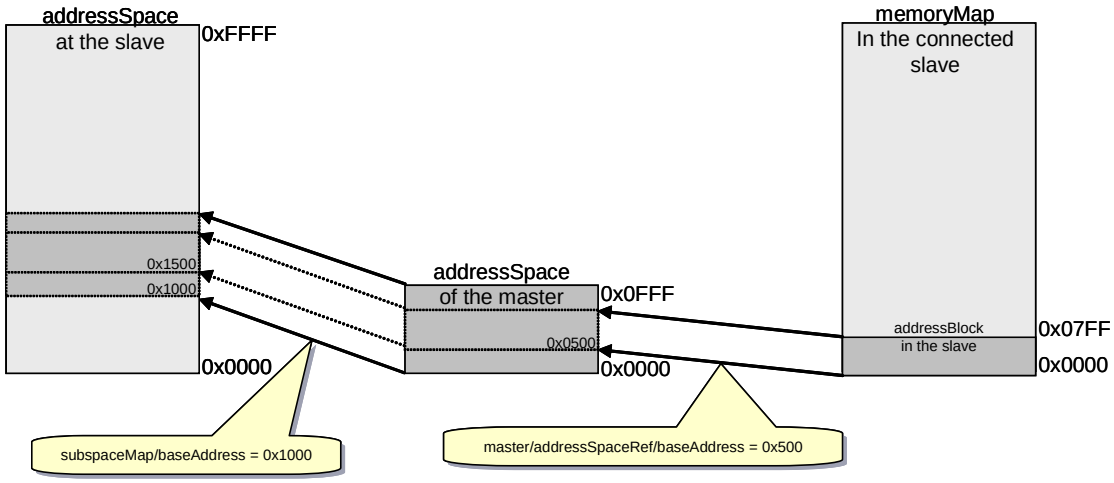


Figure H.9—Offsetting an address block in an opaque bridge

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Figure H.10 shows it is also possible to offset the **addressBlock** from the connected slave in the negative direction. The **addressBlock** from the connected slave range `0x7000` to `0x77FF` maps to the address range `0x0500` to `0x0CFF` (offset by **master**/**addressSpaceRef**/**baseAddress** = `-0x6B00`) as seen in the address space of the master interface, and to the address range `0x1500` to `0x1CFF` (offset by **subspaceMap**/**baseAddress** = `0x1000`) as seen in the address space at the slave interface.

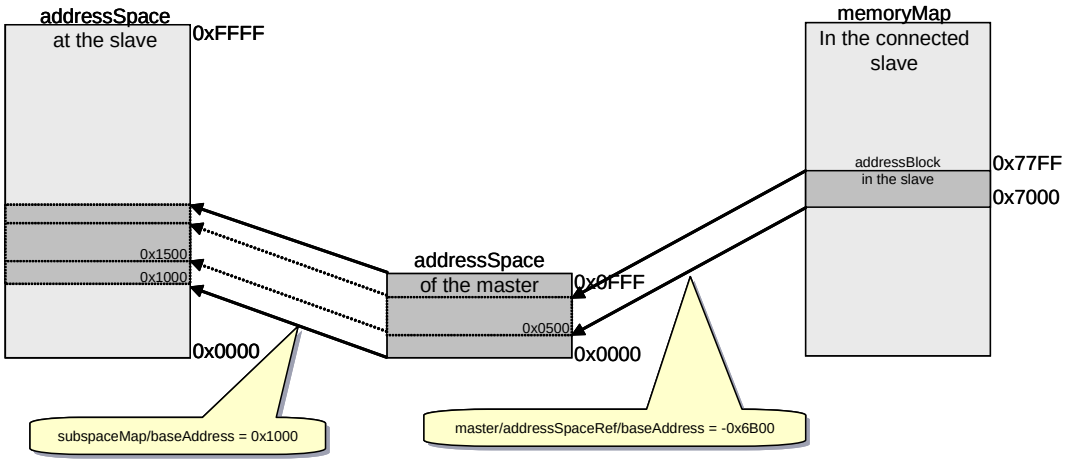


Figure H.10—Negative offsetting of an address block in an opaque bridge

Figure H.11 shows the references between the various elements and attributes in an opaque bridge.

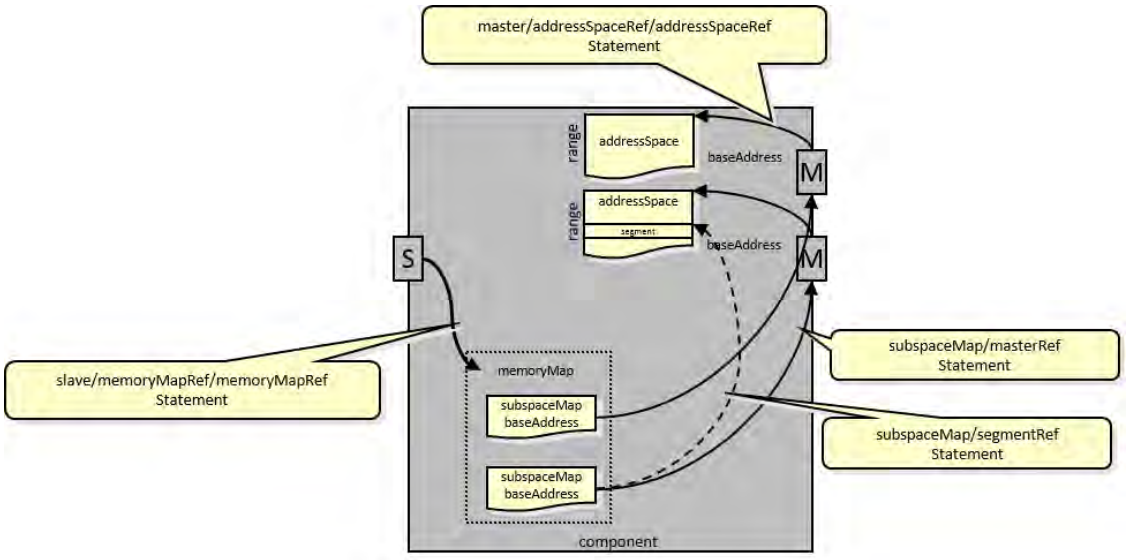


Figure H.11—Opaque bridge references

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

H.3 Channel with address remapping

A mirrored-slave interface that is part of a channel may provide a remap address for the connected slave interface. This remap address is an offset of the base address of the address block in the connected slave interface, as shown in Figure H.12. This offset is the addition the **remapAddress** element's value (see 6.10) to the base address of the memory map from the slave. The **range** element also modifies (and potentially narrows) the range of the entire memory map of the connected slave. In Figure H.12, the slave interface address range `0x0000` to `0x0FFF` maps to the address range `0x1000` to `0x17FF` (offset by **mirrorSlave**/**baseAddress**/**remapAddress** = `0x1000` and narrowed by **mirrorSlave**/**baseAddress**/**range** = `0x0800`) in the address space as seen at the mirrored-slave interface.

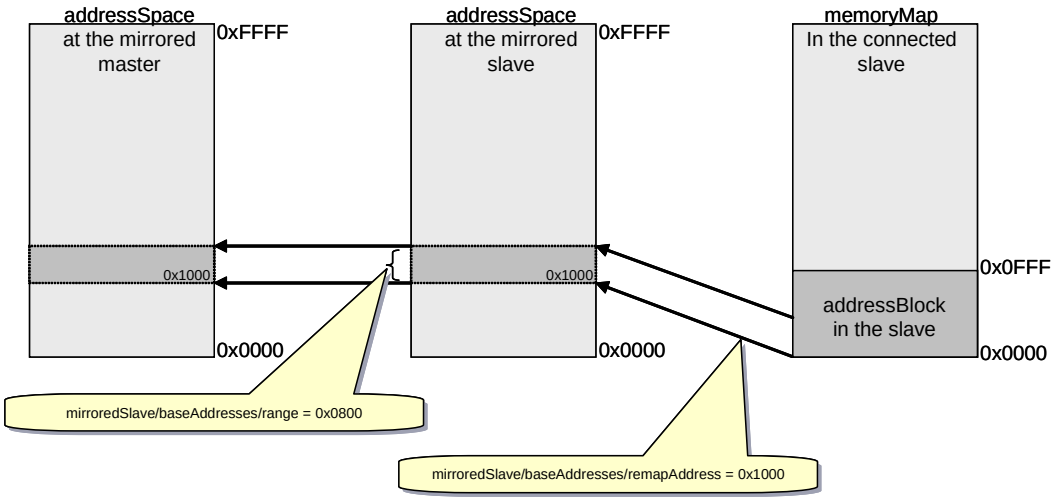


Figure H.12—Address remapping in a channel

Figure H.13 shows the references between the various elements and attributes in a channel with address remapping.

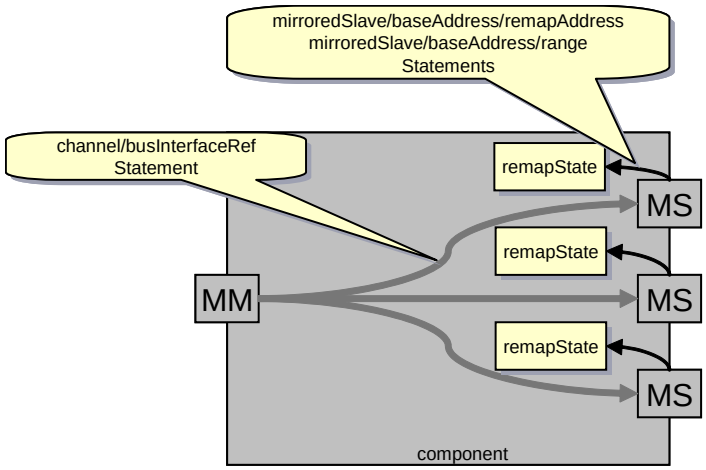


Figure H.13—Channel with remapping references

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

H.4 Channel with bit steering

A mirrored-slave interface that is part of a channel may have a **bitSteering** element. If the **bitSteering** element is **on**, the base address and range of an address block as seen across a channel is modified only by the ratio of the **bitsInLau** of the mirrored-slave and the mirrored-master interfaces. In Figure H.14, the slave interface address range `0x0000` to `0x07FF` maps to the address range `0x1000` to `0x17FF` (offset by **mirrorSlave/baseAddress/remapAddress** = `0x1000`) in the address space as seen at the mirrored-slave interface, and maps to the address range `0x1000` to `0x17FF` (multiplied by the ratio of the mirrored-slave and mirrored-master **bitsInLau** `8/8 = 1`, and independent of the width of the logical data ports) in the address space as seen at the mirrored-master interface.

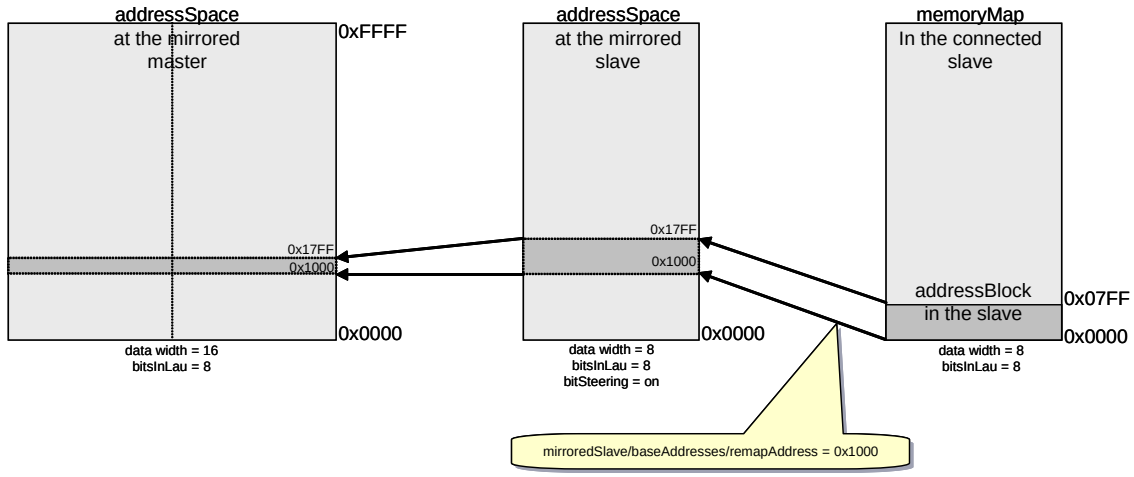


Figure H.14—Channel with equal bitsInLau and bitSteering = on

In Figure H.15, the **bitSteering** element is **on** and there are differing **bitsInLau** across the channel. The slave interface address range `0x0000` to `0x0FFF` maps to the address range `0x1000` to `0x1FFF` (offset by **mirrorSlave/baseAddress/remapAddress** = `0x1000`) in the address space as seen at the mirrored-slave interface, and maps to the address range `0x0800` to `0x0FFF` (multiplied by the ratio of the mirrored-slave and mirrored-master **bitsInLau** `4/8 = 1/2`, and independent of the width of the logical data ports) in the address space as seen at the mirrored-master interface.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

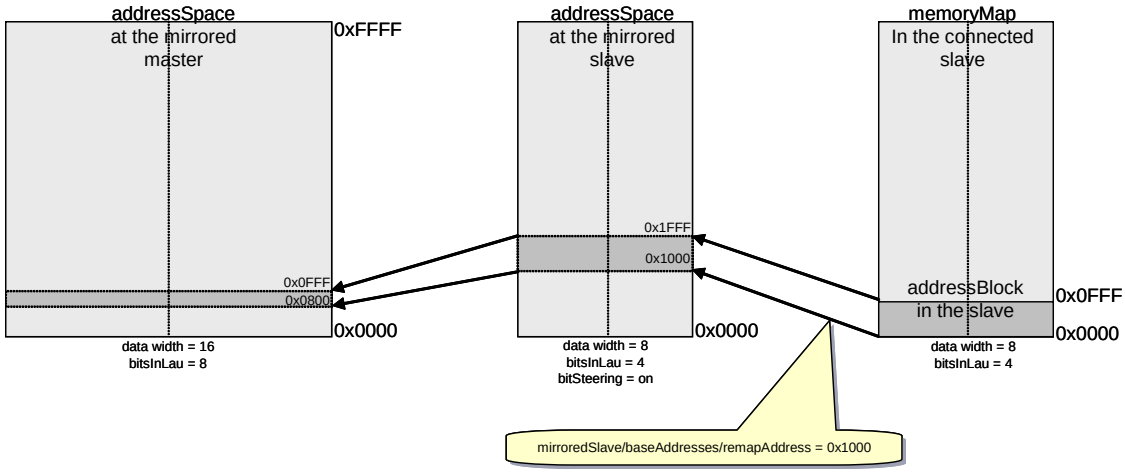


Figure H.15—Channel with non-equal bitsInLau and bitSteering = on

If **bitSteering** is **off**, the base address and range of an address block as seen across a channel is modified by the ratio of the **bitsInLau** and the logical data-width of the mirrored-slave and mirrored-master interfaces. In Figure H.16, the slave interface address range `0x0000` to `0x07FF` maps to the address range `0x1000` to `0x17FF` (offset by **mirrorSlave**/**baseAddress**/**remapAddress** = `0x1000`) in the address space as seen at the mirrored-slave interface, and maps to the address range `0x2000` to `0x2FFF` (multiplied by the ratio of mirrored-slave and mirrored-master **bitsInLau** `8/8 = 1`, and multiplied by the ratio mirrored-master logical data-width divided by the mirrored-slave logical data-width `16/8 = 2`) in the address space as seen at the mirrored-master interface.

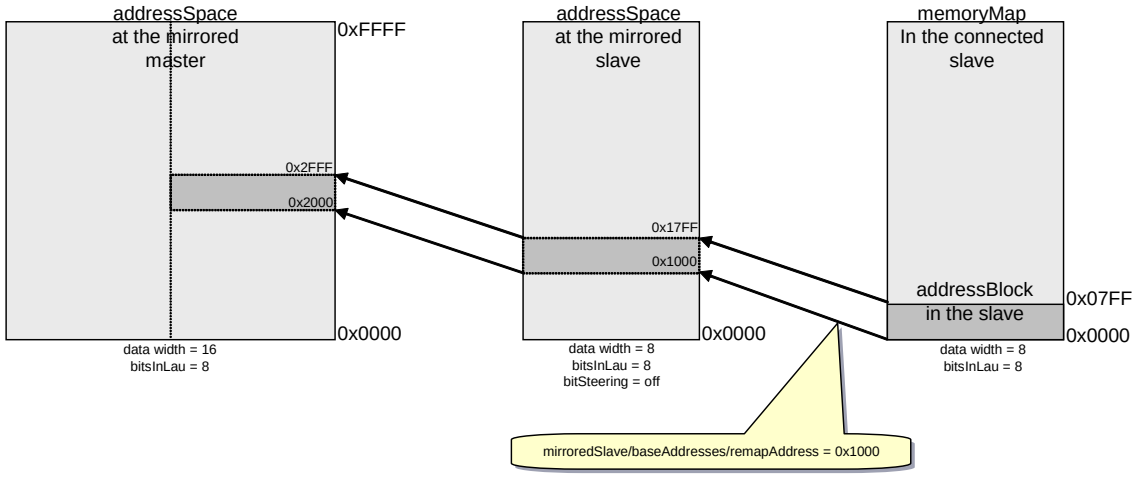


Figure H.16—Channel with bitSteering = off

Figure H.17 shows the references between the various elements and attributes in a channel with **bitSteering** references.

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

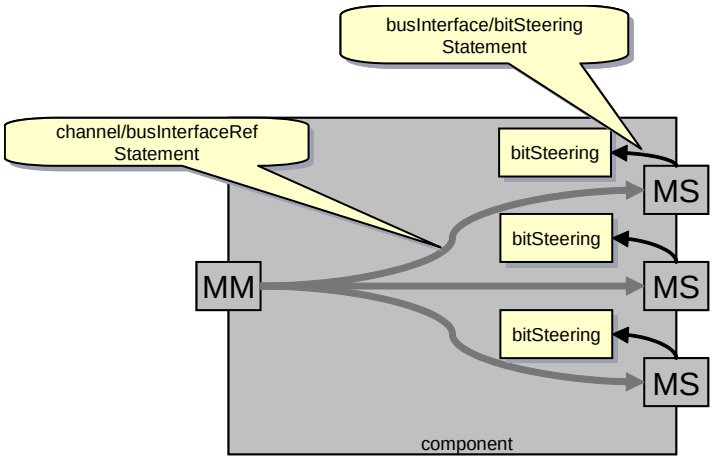


Figure H.17—Channel with bitSteering references

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Annex I

(informative)

Examples

This annex shows a set of examples [B8] covering some practical uses of IP-XACT (syntax) and a series of pointers to the various syntax definitions, as appropriate. Within each example, any `LINK:` comments show the relevant cross-reference to that element's image/descriptions.

I.1 abstractionDefinition - RTL

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
// Example abstraction definition used to show schema elements defined by the
// IP-XACT standard. Links within this file refer to schema figures in
// the standard definition document. This file represents an RTL abstraction.
-->
<!-- LINK: abstractionDefinition: see 5.3, Abstraction definition -->
<ipxact:abstractionDefinition xmlns:xsi="http://www.w3.org/2001/XMLSchema-
   instance" xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/2.0"
   xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/2.0/
   index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>Sample</ipxact:library>
  <ipxact:name>SampleAbstractionDefinition_RTL</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
  <ipxact:busType vendor="accellera.org" library="Sample"
   name="SampleBusDefinitionExtension" version="1.0"/>
  <ipxact:ports>
    <!-- LINK: port: see 5.4, Ports -->
    <!-- Simple wire port -->
    <ipxact:port>
      <ipxact:logicalName>Data</ipxact:logicalName>
      <ipxact:displayName>Data Bus</ipxact:displayName>
      <ipxact:description>The data bus</ipxact:description>
      <!-- LINK: wire: see 5.5, Wire ports -->
      <ipxact:wire>
        <!-- LINK: qualifier: see 5.6, Qualifiers -->
        <ipxact:qualifier>
          <ipxact:isData>true</ipxact:isData>
        </ipxact:qualifier>
        <ipxact:onMaster>
          <!-- LINK: wirePort: see 5.7, Wire port group -->
          <ipxact:presence>required</ipxact:presence>
          <ipxact:width>8</ipxact:width>
          <ipxact:direction>out</ipxact:direction>
          <!-- LINK: modeConstraints: see 5.8, Wire port mode (and mirrored
   mode) constraints -->
          <ipxact:modeConstraints>
            <ipxact:timingConstraint clockName="Clk">40</
   ipxact:timingConstraint>
          </ipxact:modeConstraints>
        </ipxact:onMaster>
        <ipxact:onSlave>
```

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
        <ipxact:width>8</ipxact:width>
        <ipxact:direction>in</ipxact:direction>
      </ipxact:onSlave>
      <ipxact:defaultValue>'ff</ipxact:defaultValue>
    </ipxact:wire>
  </ipxact:port>
  <ipxact:port>
    <ipxact:logicalName>Address</ipxact:logicalName>
    <ipxact:displayName>Address Bus</ipxact:displayName>
    <ipxact:description>The address bus</ipxact:description>
    <!-- LINK: wire: see 5.5, Wire ports -->
    <ipxact:wire>
      <!-- LINK: qualifier: see 5.6, Qualifiers -->
      <ipxact:qualifier>
        <ipxact:isAddress>true</ipxact:isAddress>
      </ipxact:qualifier>
      <ipxact:onMaster>
        <!-- LINK: wirePort: see 5.7, Wire port group -->
        <ipxact:presence>required</ipxact:presence>
        <ipxact:direction>out</ipxact:direction>
      </ipxact:onMaster>
      <ipxact:onSlave>
        <ipxact:width>8</ipxact:width>
        <ipxact:direction>in</ipxact:direction>
      </ipxact:onSlave>
      <ipxact:defaultValue>'ff</ipxact:defaultValue>
    </ipxact:wire>
  </ipxact:port>
  <!-- Simple clock port to show system elements -->
  <ipxact:port>
    <ipxact:logicalName>Clk</ipxact:logicalName>
    <ipxact:wire>
      <ipxact:qualifier>
        <ipxact:isClock>true</ipxact:isClock>
      </ipxact:qualifier>
      <ipxact:onSystem>
        <ipxact:group>SystemSignals</ipxact:group>
        <ipxact:width>1</ipxact:width>
        <ipxact:direction>out</ipxact:direction>
      </ipxact:onSystem>
      <ipxact:onMaster>
        <ipxact:presence>optional</ipxact:presence>
        <ipxact:width>1</ipxact:width>
        <ipxact:direction>in</ipxact:direction>
      </ipxact:onMaster>
      <ipxact:onSlave>
        <ipxact:presence>optional</ipxact:presence>
        <ipxact:width>1</ipxact:width>
        <ipxact:direction>in</ipxact:direction>
      </ipxact:onSlave>
      <ipxact:requiresDriver driverType="singleShot">true</
ipxact:requiresDriver>
    </ipxact:wire>
  </ipxact:port>
  <!-- Conditionally existing logical port -->
  <ipxact:port>
    <ipxact:isPresent>UsingParity</ipxact:isPresent>
    <ipxact:logicalName>Parity</ipxact:logicalName>
    <ipxact:wire>
```


Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
        <ipxact:onMaster>
          <ipxact:presence>required</ipxact:presence>
          <ipxact:width>1</ipxact:width>
          <ipxact:direction>out</ipxact:direction>
    </ipxact:onMaster>
        <ipxact:onSlave>
          <ipxact:presence>required</ipxact:presence>
          <ipxact:width>1</ipxact:width>
          <ipxact:direction>in</ipxact:direction>
    </ipxact:onSlave>
       </ipxact:wire>
     </ipxact:port>
  </ipxact:ports>
  <ipxact:description>Example RTL abstraction definition used in the IP-XACT
    standard.</ipxact:description>
  <ipxact:parameters>
    <ipxact:parameter parameterId="UsingParity" type="bit"
    resolve="generated">
       <ipxact:name>UsingParity</ipxact:name>
       <ipxact:description>Set to true if parity interface being used.</
    ipxact:description>
       <ipxact:value>0</ipxact:value>
     </ipxact:parameter>
  </ipxact:parameters>
</ipxact:abstractionDefinition>
```

I.2 abstractionDefinition - TLM

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
// Example abstraction definition used to show schema elements defined by the
// IP-XACT standard. Links within this file refer to schema figures in
// the standard definition document. This is a trivial TLM abstraction.
-->
<!-- LINK: abstractionDefinition: see 5.3, Abstraction definition -->
<ipxact:abstractionDefinition xmlns:xsi="http://www.w3.org/2001/XMLSchema-
   instance" xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/2.0"
   xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/2.0/
   index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>Sample</ipxact:library>
  <ipxact:name>SampleAbstractionDefinition_TLM</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
  <ipxact:busType vendor="accellera.org" library="Sample"
   name="SampleBusDefinitionExtension" version="1.0"/>
  <ipxact:ports>
    <!-- LINK: port: see 5.4, Ports -->
    <!-- Simple transactional port -->
    <ipxact:port>
      <ipxact:logicalName>Transport</ipxact:logicalName>
      <!-- LINK: transactional: see 5.9, Transactional ports -->
      <ipxact:transactional>
        <ipxact:onMaster>
          <!-- abstractionDefinition/transactionalPort -->
          <ipxact:presence>optional</ipxact:presence>
          <ipxact:initiative>provides</ipxact:initiative>
          <ipxact:kind>simple_socket</ipxact:kind>
```

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
            <ipxact:busWidth>8</ipxact:busWidth>
            <!-- LINK: protocol: see 6.12.8, Component transactional protocol/
      payload definition -->
            <ipxact:protocol>
              <ipxact:protocolType>tlm</ipxact:protocolType>
              <ipxact:payload>
                <ipxact:name>tlm2</ipxact:name>
                <ipxact:type>generic</ipxact:type>
              </ipxact:payload>
            </ipxact:protocol>
          </ipxact:onMaster>
          <ipxact:onSlave>
            <ipxact:initiative>requires</ipxact:initiative>
            <ipxact:kind>simple_socket</ipxact:kind>
            <ipxact:protocol>
              <ipxact:protocolType>tlm</ipxact:protocolType>
              <ipxact:payload>
                <ipxact:name>tlm2</ipxact:name>
                <ipxact:type>generic</ipxact:type>
              </ipxact:payload>
            </ipxact:protocol>
          </ipxact:onSlave>
        </ipxact:transactional>
      </ipxact:port>
    </ipxact:ports>
    <ipxact:description>Example TLM abstraction definition used in the IP-XACT
     standard.</ipxact:description>
  </ipxact:abstractionDefinition>
```

I.3 abstractor

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
// Example abstractor used to show schema elements defined by the
// IP-XACT standard. Links within this file refer to schema figures in
// the standard definition document.
-->
<!-- LINK: abstractor: see 8.1, Abstractor -->
<ipxact:abstractor xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/2.0"
    xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/2.0/
    index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>Sample</ipxact:library>
  <ipxact:name>SampleAbstractor</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
  <ipxact:abstractorMode>master</ipxact:abstractorMode>
  <ipxact:busType vendor="accellera.org" library="Sample"
   name="SampleBusDefinition" version="1.0"/>
  <!-- LINK: abstractorInterfaces: see 8.2, Abstractor interfaces -->
  <ipxact:abstractorInterfaces>
    <!-- master interface -->
    <ipxact:abstractorInterface>
      <ipxact:name>Master</ipxact:name>
      <ipxact:abstractionTypes>
        <ipxact:abstractionType>
```


Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
          <ipxact:abstractionRef vendor="accellera.org" library="Sample"
    name="SampleAbstractionDefinition_TLM" version="1.0"/>
        </ipxact:abstractionType>
      </ipxact:abstractionTypes>
    </ipxact:abstractorInterface>
    <!-- mirrored master interface -->
    <ipxact:abstractorInterface>
      <ipxact:name>MirroredMaster</ipxact:name>
      <ipxact:abstractionTypes>
        <ipxact:abstractionType>
          <ipxact:abstractionRef vendor="accellera.org" library="Sample"
    name="SampleAbstractionDefinition_RTL" version="1.0"/>
        </ipxact:abstractionType>
      </ipxact:abstractionTypes>
    </ipxact:abstractorInterface>
  </ipxact:abstractorInterfaces>

  <!-- Remaining content in abstractor is optional and similar enough -->
  <!-- to a component that it is excluded from the example -->

</ipxact:abstractor>
```

I.4 busDefinition

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
// Example bus definition used to show schema elements defined by the
// IP-XACT standard. Links within this file refer to schema figures in
// the standard definition document
-->
<!-- LINK: busDefinition/busDefinition: see 5.2, Bus definition -->
<ipxact:busDefinition xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/2.0"
    xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/2.0/
   index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>Sample</ipxact:library>
  <ipxact:name>SampleBusDefinitionExtension</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
  <ipxact:directConnection>true</ipxact:directConnection>
  <ipxact:broadcast>true</ipxact:broadcast>
  <ipxact:isAddressable>false</ipxact:isAddressable>
  <ipxact:extends vendor="accellera.org" library="Sample"
            name="SampleBusDefinitionBase" version="1.0"/>
  <ipxact:maxMasters>1</ipxact:maxMasters>
  <ipxact:maxSlaves>16</ipxact:maxSlaves>
  <ipxact:systemGroupNames>
    <ipxact:systemGroupName>SystemSignals</ipxact:systemGroupName>
  </ipxact:systemGroupNames>
  <ipxact:description>Example bus definition used in the IP-XACT standard.</
   ipxact:description>
  <!-- ipxact:parameters - excluded for brevity - see component example -->
  <!-- ipxact:assertions - excluded for brevity - see component example -->
  <!-- ipxact:vendorExtensions - excluded for brevity - see component example
    -->
</ipxact:busDefinition>
```

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

I.5 catalog

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
// Example catalog used to show schema elements defined by the
// IP-XACT standard. Links within this file refer to schema figures in
// the standard definition document
-->
<!-- LINK: catalog: see 11.1, catalog -->
<ipxact:catalog xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/2.0"
    xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/2.0/
    index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>Sample</ipxact:library>
  <ipxact:name>SampleCatalog</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
  <ipxact:description>Example catalog used in the IP-XACT standard.</
   ipxact:description>
  <ipxact:busDefinitions>
    <!-- LINK: catalog/ipxactFile: see 11.2, ipxactFile -->
    <ipxact:ipxactFile>
      <ipxact:vlnv vendor="accellera.org" library="Sample"
   name="SampleBusDefinitionBase" version="1.0"/>
      <ipxact:name>./SampleBusDefinitionBase.xml</ipxact:name>
      <ipxact:description>File included for semantic validation only.</
   ipxact:description>
    </ipxact:ipxactFile>
    <ipxact:ipxactFile>
      <ipxact:vlnv vendor="accellera.org" library="Sample"
   name="SampleBusDefinitionExtension" version="1.0"/>
      <ipxact:name>./SampleBusDefinitionExtension.xml</ipxact:name>
      <ipxact:description>Sample extended busDefinition</ipxact:description>
    </ipxact:ipxactFile>
  </ipxact:busDefinitions>
  <ipxact:abstractionDefinitions>
    <ipxact:ipxactFile>
      <ipxact:vlnv vendor="accellera.org" library="Sample"
   name="SampleAbstractionDefinition_RTL" version="1.0"/>
      <ipxact:name>./SampleAbstractionDefinition_RTL.xml</ipxact:name>
    </ipxact:ipxactFile>
    <ipxact:ipxactFile>
      <ipxact:vlnv vendor="accellera.org" library="Sample"
   name="SampleAbstractionDefinition_TLM" version="1.0"/>
      <ipxact:name>./SampleAbstractionDefinition_TLM.xml</ipxact:name>
    </ipxact:ipxactFile>
  </ipxact:abstractionDefinitions>
  <!-- ipxact:components -->
  <ipxact:components>
    <ipxact:ipxactFile>
      <ipxact:vlnv vendor="accellera.org" library="Sample"
   name="SampleComponent" version="1.0"/>
      <ipxact:name>./SampleComponent.xml</ipxact:name>
    </ipxact:ipxactFile>
  </ipxact:components>
  <!-- ipxact:abstractors -->
  <ipxact:abstractors>
    <ipxact:ipxactFile>
```


Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
      <ipxact:vlnv vendor="accellera.org" library="Sample"
    name="SampleAbstractor" version="1.0"/>
      <ipxact:name>./SampleAbstractor.xml</ipxact:name>
     </ipxact:ipxactFile>
  </ipxact:abstractors>
  <!-- ipxact:designs -->
  <ipxact:designs>
    <ipxact:ipxactFile>
     <ipxact:vlnv vendor="accellera.org" library="Sample" name="SampleDesign"
    version="1.0"/>
      <ipxact:name>./SampleDesign.xml</ipxact:name>
     </ipxact:ipxactFile>
  </ipxact:designs>
  <!-- ipxact:designConfigurations -->
  <ipxact:designConfigurations>
    <ipxact:ipxactFile>
      <ipxact:vlnv vendor="accellera.org" library="Sample"
    name="SampleDesignConfiguration" version="1.0"/>
      <ipxact:name>./SampleDesignConfiguration.xml</ipxact:name>
     </ipxact:ipxactFile>
  <ipxact:designConfigurations>
  <ipxact:generatorChains>
    <ipxact:ipxactFile>
      <ipxact:vlnv vendor="accellera.org" library="Sample"
    name="SampleGeneratorChain" version="1.0"/>
      <ipxact:name>./SampleGeneratorChain.xml</ipxact:name>
     </ipxact:ipxactFile>
     <ipxact:ipxactFile>
      <ipxact:vlnv vendor="accellera.org" library="Sample"
    name="SampleGeneratorChainForReference" version="1.0"/>
      <ipxact:name>./SampleGeneratorChainForReference.xml</ipxact:name>
     </ipxact:ipxactFile>
  </ipxact:generatorChains>
</ipxact:catalog>
```

I.6 component

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
// Example component used to show schema elements defined by the
// IP-XACT standard. Links within this file refer to schema figures in
// the standard definition document.
-->
<!-- LINK: component: see 6.1, Component -->
<ipxact:component xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/2.0"
   xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/2.0/
   index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
  <ipxact:library>Sample</ipxact:library>
  <ipxact:name>SampleComponent</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
  <ipxact:busInterfaces>
    <!-- LINK: busInterface: 6.5.1, busInterface -->
    <!-- Basic slave interface with both RTL and TLM representations -->
    <ipxact:busInterface>
      <ipxact:name>Slave</ipxact:name>
```

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
    <ipxact:busType vendor="accellera.org" library="Sample"
name="SampleBusDefinition" version="1.0"/>
    <!-- LINK: abstractionTypes: 6.5.6, Abstraction types -->
    <ipxact:abstractionTypes>
      <ipxact:abstractionType>
        <ipxact:viewRef>RTLview</ipxact:viewRef>
        <ipxact:abstractionRef vendor="accellera.org" library="Sample"
name="SampleAbstractionDefinition_RTL" version="1.0">
          <!-- Configure abstraction to have the 'Parity' logical port -->
          <ipxact:configurableElementValues>
            <ipxact:configurableElementValue referenceId="UsingParity">1
            </ipxact:configurableElementValue>
          </ipxact:configurableElementValues>
        </ipxact:abstractionRef>
        <ipxact:portMaps>
          <!-- LINK: portMap: 6.5.7, Port map -->
          <ipxact:portMap>
            <ipxact:logicalPort>
              <ipxact:name>Data</ipxact:name>
            </ipxact:logicalPort>
            <ipxact:physicalPort>
              <ipxact:name>slv_data</ipxact:name>
              <ipxact:partSelect>
                <ipxact:range>
                  <ipxact:left>7</ipxact:left>
                  <ipxact:right>0</ipxact:right>
                </ipxact:range>
              </ipxact:partSelect>
            </ipxact:physicalPort>
          </ipxact:portMap>
          <ipxact:portMap>
            <ipxact:logicalPort>
              <ipxact:name>Address</ipxact:name>
            </ipxact:logicalPort>
            <ipxact:physicalPort>
              <ipxact:name>slv_addr</ipxact:name>
            </ipxact:physicalPort>
          </ipxact:portMap>
          <ipxact:portMap>
            <ipxact:logicalPort>
              <ipxact:name>Parity</ipxact:name>
            </ipxact:logicalPort>
            <ipxact:physicalPort>
              <ipxact:name>slv_parity</ipxact:name>
            </ipxact:physicalPort>
          </ipxact:portMap>
          <!-- Mapping to clock port included for 'information' only - not
for connecting -->
          <ipxact:portMap>
            <ipxact:logicalPort>
              <ipxact:name>Clk</ipxact:name>
            </ipxact:logicalPort>
            <ipxact:physicalPort>
              <ipxact:name>clk</ipxact:name>
            </ipxact:physicalPort>
            <ipxact:isInformative>true</ipxact:isInformative>
          </ipxact:portMap>
        </ipxact:portMaps>
      </ipxact:abstractionType>
```


Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
    <ipxact:abstractionType>
      <ipxact:viewRef>TLMview</ipxact:viewRef>
      <ipxact:abstractionRef vendor="accellera.org" library="Sample"
name="SampleAbstractionDefinition_TLM" version="1.0"/>
      <ipxact:portMaps>
        <ipxact:portMap>
          <ipxact:logicalPort>
            <ipxact:name>Transport</ipxact:name>
          </ipxact:logicalPort>
          <ipxact:physicalPort>
            <ipxact:name>slv_transaction</ipxact:name>
          </ipxact:physicalPort>
        </ipxact:portMap>
      </ipxact:portMaps>
    </ipxact:abstractionType>
  </ipxact:abstractionTypes>
  <!-- LINK: interfaceMode: see 6.5.2, Interface modes -->
  <!-- LINK: slave: see 6.5.4, Slave interface -->
  <ipxact:slave>
    <ipxact:memoryMapRef memoryMapRef="SimpleMapWithBlock"/>
  </ipxact:slave>
  <!-- This interface must be connected in any containing design -->
  <ipxact:connectionRequired>true</ipxact:connectionRequired>
</ipxact:busInterface>
<!-- Basic master interface with both RTL and TLM representations -->
<ipxact:busInterface>
  <ipxact:name>Master</ipxact:name>
  <ipxact:busType vendor="accellera.org" library="Sample"
name="SampleBusDefinition" version="1.0"/>
  <ipxact:abstractionTypes>
    <ipxact:abstractionType>
      <ipxact:viewRef>RTLview</ipxact:viewRef>
      <ipxact:abstractionRef vendor="accellera.org" library="Sample"
name="SampleAbstractionDefinition_RTL" version="1.0">
       <!-- Configure abstraction to not have the 'Parity' logical port -->
        <!-- This is the default but included to make it explicit -->
        <ipxact:configurableElementValues>
          <ipxact:configurableElementValue referenceId="UsingParity">0
          </ipxact:configurableElementValue>
        </ipxact:configurableElementValues>
      </ipxact:abstractionRef>
      <ipxact:portMaps>
        <ipxact:portMap>
          <ipxact:logicalPort>
            <ipxact:name>Data</ipxact:name>
          </ipxact:logicalPort>
          <ipxact:physicalPort>
            <ipxact:name>mst_data</ipxact:name>
          </ipxact:physicalPort>
        </ipxact:portMap>
        <ipxact:portMap>
          <ipxact:logicalPort>
            <ipxact:name>Address</ipxact:name>
          </ipxact:logicalPort>
          <ipxact:physicalPort>
            <ipxact:name>mst_addr</ipxact:name>
          </ipxact:physicalPort>
        </ipxact:portMap>
```

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
        <!-- Mapping to clock port included for 'information' only - not
for connecting -->
        <ipxact:portMap>
          <ipxact:logicalPort>
            <ipxact:name>Clk</ipxact:name>
          </ipxact:logicalPort>
          <ipxact:physicalPort>
            <ipxact:name>clk</ipxact:name>
          </ipxact:physicalPort>
          <ipxact:isInformative>true</ipxact:isInformative>
        </ipxact:portMap>
      </ipxact:portMaps>
    </ipxact:abstractionType>
    <ipxact:abstractionType>
      <ipxact:viewRef>TLMview</ipxact:viewRef>
      <ipxact:abstractionRef vendor="accellera.org" library="Sample"
name="SampleAbstractionDefinition_TLM" version="1.0"/>
      <ipxact:portMaps>
        <ipxact:portMap>
          <ipxact:logicalPort>
            <ipxact:name>Transport</ipxact:name>
          </ipxact:logicalPort>
          <ipxact:physicalPort>
            <ipxact:name>mst_transaction</ipxact:name>
          </ipxact:physicalPort>
        </ipxact:portMap>
      </ipxact:portMaps>
    </ipxact:abstractionType>
  </ipxact:abstractionTypes>
  <!-- LINK: master: see 6.5.3, Master interface -->
  <ipxact:master>
    <ipxact:addressSpaceRef addressSpaceRef="simpleAddressSpace"/>
  </ipxact:master>
 </ipxact:busInterface>
 <ipxact:busInterface>
  <ipxact:name>MirroredMaster</ipxact:name>
  <ipxact:busType vendor="accellera.org" library="Sample"
name="SampleBusDefinition" version="1.0"/>
  <ipxact:mirroredMaster/>
 </ipxact:busInterface>
 <ipxact:busInterface>
  <ipxact:name>MirroredSlave</ipxact:name>
  <ipxact:busType vendor="accellera.org" library="Sample"
name="SampleBusDefinition" version="1.0"/>
  <!-- LINK: master: see 6.5.5, Mirrored slave interface -->
  <ipxact:mirroredSlave>
    <ipxact:baseAddresses>
      <ipxact:remapAddress>0x40000000</ipxact:remapAddress>
      <ipxact:range>0x1000</ipxact:range>
    </ipxact:baseAddresses>
  </ipxact:mirroredSlave>
 </ipxact:busInterface>
 <ipxact:busInterface>
  <ipxact:name>SlaveWithTransparentBridge</ipxact:name>
  <ipxact:busType vendor="accellera.org" library="Sample"
name="SampleBusDefinition" version="1.0"/>
  <ipxact:slave>
    <ipxact:transparentBridge masterRef="Master"/>
  </ipxact:slave>
```


Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
  </ipxact:busInterface>
  <ipxact:busInterface>
    <ipxact:name>SlaveForSubspaceMap</ipxact:name>
    <ipxact:busType vendor="accellera.org" library="Sample"
 name="SampleBusDefinition" version="1.0"/>
    <ipxact:slave>
      <ipxact:memoryMapRef memoryMapRef="SimpleMapWithSubspace"/>
    </ipxact:slave>
  </ipxact:busInterface>
  <ipxact:busInterface>
    <ipxact:name>Monitor</ipxact:name>
    <ipxact:busType vendor="accellera.org" library="Sample"
 name="SampleBusDefinition" version="1.0"/>
    <ipxact:monitor interfaceMode="slave"/>
  </ipxact:busInterface>
</ipxact:busInterfaces>
<ipxact:indirectInterfaces>
  <!-- LINK: indirectInterface: see 6.6, Indirect interfaces -->
  <ipxact:indirectInterface>
    <ipxact:name>IndirectMemoryInterface</ipxact:name>
    <ipxact:indirectAddressRef>IAR</ipxact:indirectAddressRef>
    <ipxact:indirectDataRef>IDR</ipxact:indirectDataRef>
    <ipxact:memoryMapRef>MapForMemory</ipxact:memoryMapRef>
  </ipxact:indirectInterface>
</ipxact:indirectInterfaces>
<ipxact:channels>
  <!-- LINK: channel: see 6.7, Component channels -->
  <ipxact:channel>
    <ipxact:name>theChannel</ipxact:name>
    <ipxact:busInterfaceRef>
      <ipxact:localName>MirroredMaster</ipxact:localName>
    </ipxact:busInterfaceRef>
    <ipxact:busInterfaceRef>
      <ipxact:localName>MirroredSlave</ipxact:localName>
    </ipxact:busInterfaceRef>
  </ipxact:channel>
</ipxact:channels>
<ipxact:remapStates>
  <!-- LINK: remapStates: see 6.10.2, Remap states-->
  <ipxact:remapState>
    <ipxact:name>Boot</ipxact:name>
   <ipxact:description>Defines the remap state associated with the component
 at boot time.</ipxact:description>
  </ipxact:remapState>
  <ipxact:remapState>
    <ipxact:name>Nornmal</ipxact:name>
   <ipxact:description>Defines the remap state associated with the component
 after boot time.</ipxact:description>
  </ipxact:remapState>
</ipxact:remapStates>
<ipxact:addressSpaces>
  <!-- LINK: addressSpace: see 6.8.1, addressSpaces -->
  <ipxact:addressSpace>
    <ipxact:name>simpleAddressSpace</ipxact:name>
    <ipxact:range>4*(2**30)</ipxact:range>
    <ipxact:width>32</ipxact:width>
  </ipxact:addressSpace>
</ipxact:addressSpaces>
<ipxact:memoryMaps>
```

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
<!-- LINK: memoryMap: see 6.9.1, memoryMaps -->
<ipxact:memoryMap>
  <ipxact:name>SimpleMapWithBlock</ipxact:name>
  <!-- Address block starts at address 0, containing 1024 addressable 8-
bit -->
  <!-- units, organized into larger 32-bit units. -->
  <!-- LINK: addressBlock: see 6.9.2, Address block -->
  <ipxact:addressBlock>
    <ipxact:name>SimpleAddressBlock</ipxact:name>
    <ipxact:baseAddress>0x0</ipxact:baseAddress>
    <!-- LINK: addressBlockDefinitionGroup: see 6.9.3, Address block
definition group -->
    <ipxact:range>2**10</ipxact:range>
    <ipxact:width>32</ipxact:width>
    <!-- LINK: memoryBlockData: see 6.9.4, memoryBlockData group -->
    <ipxact:usage>register</ipxact:usage>
    <ipxact:volatile>false</ipxact:volatile>
    <ipxact:access>read-write</ipxact:access>
    <!-- LINK: registerData: see 6.11.1, Register data -->
    <!-- LINK: register: see 6.11.2, Register -->
    <ipxact:register>
      <ipxact:name>BasicRegister</ipxact:name>
      <ipxact:addressOffset>0x4</ipxact:addressOffset>
      <!-- LINK: registerDefinitionGroup: see 6.11.3, Register definition
group -->
      <ipxact:size>32</ipxact:size>
      <ipxact:volatile>true</ipxact:volatile>
      <ipxact:access>read-writeOnce</ipxact:access>
      <!-- LINK: field: see 6.11.8, Register bit fields -->
      <ipxact:field>
        <ipxact:name>F1</ipxact:name>
        <ipxact:bitOffset>0</ipxact:bitOffset>
        <ipxact:resets>
          <ipxact:reset>
            <ipxact:value>0x0</ipxact:value>
          </ipxact:reset>
          <ipxact:reset resetTypeRef="SOFT">
            <ipxact:value>0xf</ipxact:value>
            <ipxact:mask>0xa</ipxact:mask>
          </ipxact:reset>
        </ipxact:resets>
        <ipxact:bitWidth>4</ipxact:bitWidth>
        <!-- LINK: fieldData: see 6.11.9 Field data group -->
        <ipxact:access>writeOnce</ipxact:access>
      <!-- Document pre-defined values that can be written to this field -->
        <ipxact:enumeratedValues>
          <!-- LINK: enumeratedValue: see 6.11.10, Enumeration values -->
          <ipxact:enumeratedValue>
            <ipxact:name>SetPos0</ipxact:name>
            <ipxact:value>0x1</ipxact:value>
          </ipxact:enumeratedValue>
          <ipxact:enumeratedValue>
            <ipxact:name>SetPos1</ipxact:name>
            <ipxact:value>0x2</ipxact:value>
          </ipxact:enumeratedValue>
        </ipxact:enumeratedValues>
      </ipxact:field>
      <ipxact:field>
        <ipxact:name>F2</ipxact:name>
```


Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
          <ipxact:bitOffset>4</ipxact:bitOffset>
          <ipxact:resets>
            <ipxact:reset>
              <ipxact:value>0x0</ipxact:value>
            </ipxact:reset>
          </ipxact:resets>
          <ipxact:bitWidth>4</ipxact:bitWidth>
          <ipxact:modifiedWriteValue>oneToClear</ipxact:modifiedWriteValue>
          <ipxact:readAction>modify</ipxact:readAction>
          <ipxact:testable testConstraint="readOnly">true</ipxact:testable>
        </ipxact:field>
        <ipxact:field>
          <ipxact:name>F3</ipxact:name>
          <ipxact:bitOffset>8</ipxact:bitOffset>
          <ipxact:bitWidth>4</ipxact:bitWidth>
         <!-- LINK: writeValueConstraint: see 6.11.11, Write value constraint
-->
          <ipxact:writeValueConstraint>
            <ipxact:minimum>0x1</ipxact:minimum>
            <ipxact:maximum>0x2</ipxact:maximum>
          </ipxact:writeValueConstraint>
        </ipxact:field>
        <ipxact:field>
          <ipxact:name>F4</ipxact:name>
          <ipxact:bitOffset>12</ipxact:bitOffset>
          <ipxact:bitWidth>20</ipxact:bitWidth>
          <ipxact:reserved>true</ipxact:reserved>
        </ipxact:field>
      </ipxact:register>
      <ipxact:register>
        <ipxact:name>IAR</ipxact:name>
        <ipxact:addressOffset>0x8</ipxact:addressOffset>
        <ipxact:size>32</ipxact:size>
        <ipxact:field fieldID="IAR">
          <ipxact:name>IAR</ipxact:name>
          <ipxact:bitOffset>0</ipxact:bitOffset>
          <ipxact:bitWidth>32</ipxact:bitWidth>
        </ipxact:field>
      </ipxact:register>
      <ipxact:register>
        <ipxact:name>IDR</ipxact:name>
        <ipxact:addressOffset>0xc</ipxact:addressOffset>
        <ipxact:size>32</ipxact:size>
        <ipxact:field fieldID="IDR">
          <ipxact:name>IDR</ipxact:name>
          <ipxact:bitOffset>0</ipxact:bitOffset>
          <ipxact:bitWidth>32</ipxact:bitWidth>
        </ipxact:field>
      </ipxact:register>
      <ipxact:register>
        <ipxact:name>RegisterWithAlternate</ipxact:name>
        <ipxact:addressOffset>0x10</ipxact:addressOffset>
        <ipxact:size>32</ipxact:size>
        <ipxact:access>write-only</ipxact:access>
        <ipxact:field>
          <ipxact:name>F</ipxact:name>
          <ipxact:bitOffset>0</ipxact:bitOffset>
          <ipxact:bitWidth>32</ipxact:bitWidth>
        </ipxact:field>
```

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
        <ipxact:alternateRegisters>
          <!-- LINK: alternateRegister: see 6.11.4, Alternate register -->
          <ipxact:alternateRegister>
            <ipxact:name>Alternate1</ipxact:name>
            <ipxact:alternateGroups>
              <ipxact:alternateGroup>AltRegGroup</ipxact:alternateGroup>
            </ipxact:alternateGroups>
           <!-- LINK: alternateRegisterDefinitionGroup: see 6.11.5, Alternate
register definition group -->
            <ipxact:access>read-only</ipxact:access>
            <ipxact:field>
              <ipxact:name>F</ipxact:name>
              <ipxact:bitOffset>0</ipxact:bitOffset>
              <ipxact:bitWidth>32</ipxact:bitWidth>
            </ipxact:field>
          </ipxact:alternateRegister>
        </ipxact:alternateRegisters>
      </ipxact:register>
      <!-- 8 registers combined in an array with 32 bit gap -->
      <!-- LINK: registerFile: see 6.11.6, Register file -->
      <ipxact:registerFile>
        <ipxact:name>RegisterArray</ipxact:name>
        <ipxact:dim>8</ipxact:dim>
        <ipxact:addressOffset>0x200</ipxact:addressOffset>
        <ipxact:range>0x10</ipxact:range>
        <ipxact:register>
          <ipxact:name>RegArrayEntry</ipxact:name>
          <ipxact:addressOffset>0x0</ipxact:addressOffset>
          <ipxact:size>32</ipxact:size>
          <ipxact:field>
            <ipxact:name>MandatoryField</ipxact:name>
            <ipxact:bitOffset>0</ipxact:bitOffset>
            <ipxact:bitWidth>32</ipxact:bitWidth>
          </ipxact:field>
        </ipxact:register>
      </ipxact:registerFile>
    </ipxact:addressBlock>
    <ipxact:addressUnitBits>8</ipxact:addressUnitBits>
  </ipxact:memoryMap>
  <ipxact:memoryMap>
    <ipxact:name>SimpleMapWithBank</ipxact:name>
    <!-- Serial bank with two memory blocks of 1 k units of 32-bit data. -->
   <!-- The only address specified is 0x10000, but this causes address block
-->
    <!-- ram0 and ram1 to be mapped to addresses 0x10000 and 0x11000
respectively. -->
    <!-- LINK: addressBank: see 6.9.5, Bank -->
    <ipxact:bank bankAlignment="serial">
      <ipxact:name>SerialBank</ipxact:name>
      <ipxact:baseAddress>0x1000</ipxact:baseAddress>
      <ipxact:addressBlock>
        <ipxact:name>ram0</ipxact:name>
        <ipxact:range>0x1000</ipxact:range>
        <ipxact:width>32</ipxact:width>
      </ipxact:addressBlock>
      <ipxact:addressBlock>
        <ipxact:name>ram1</ipxact:name>
        <ipxact:range>0x1000</ipxact:range>
        <ipxact:width>32</ipxact:width>
```


Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
        </ipxact:addressBlock>
      </ipxact:bank>
    </ipxact:memoryMap>
    <ipxact:memoryMap>
      <ipxact:name>SimpleMapWithSubspace</ipxact:name>
      <!-- Address space from master interface 'Master' mapped into slave
   interface -->
      <!-- SlaveForSubspaceMap at address 0x10000 -->
      <!-- LINK: subspaceMap: see 6.9.9, Subsapce map -->
      <ipxact:subspaceMap masterRef="Master">
        <ipxact:name>SubSpace</ipxact:name>
        <ipxact:baseAddress>0x10000</ipxact:baseAddress>
      </ipxact:subspaceMap>
    </ipxact:memoryMap>
    <ipxact:memoryMap>
      <ipxact:name>MapForMemory</ipxact:name>
      <ipxact:addressBlock>
        <ipxact:name>MemoryBlock</ipxact:name>
        <ipxact:baseAddress>0x0</ipxact:baseAddress>
        <ipxact:range>2**10</ipxact:range>
        <ipxact:width>32</ipxact:width>
        <ipxact:usage>memory</ipxact:usage>
        <ipxact:access>read-write</ipxact:access>
      </ipxact:addressBlock>
    </ipxact:memoryMap>
  </ipxact:memoryMaps>
<!-- LINK: model: see 6.12.1, Model -->
  <ipxact:model>
    <ipxact:views>
      <ipxact:view>
        <ipxact:name>RTLview</ipxact:name>
        <ipxact:displayName>RTL View</ipxact:displayName>
      <ipxact:description>Simple RTL view of a component.</ipxact:description>
        <ipxact:envIdentifier>:*Synthesis:</ipxact:envIdentifier>
        <ipxact:componentInstantiationRef>VerilogModel</
   ipxact:componentInstantiationRef>
      </ipxact:view>
      <ipxact:view>
        <ipxact:name>TLMview</ipxact:name>
        <ipxact:displayName>TLM View</ipxact:displayName>
      <ipxact:description>Simple TLM view of a component.</ipxact:description>
        <ipxact:isPresent>TLMModelsAvailable</ipxact:isPresent>
        <ipxact:envIdentifier>:*Simulation:</ipxact:envIdentifier>
        <ipxact:componentInstantiationRef>TLMModel</
   ipxact:componentInstantiationRef>
      </ipxact:view>
    </ipxact:views>
    <ipxact:instantiations>
      <!-- LINK: instantiationsGroup: see 6.12.2, instantiationsGroup -->
      <!-- LINK: componentInstantiation: see 6.12.3, componentInstantiation -->
      <ipxact:componentInstantiation>
        <ipxact:name>VerilogModel</ipxact:name>
        <ipxact:language>verilog</ipxact:language>
        <ipxact:moduleName>sample</ipxact:moduleName>
        <!-- LINK: moduleParameters: see 6.12.6, Module parameters -->
        <ipxact:moduleParameters>
          <ipxact:moduleParameter type="bit">
            <ipxact:name>dual_mode_reg_value</ipxact:name>
            <ipxact:vectors>
```

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
            <ipxact:vector>
              <ipxact:left>3</ipxact:left>
              <ipxact:right>0</ipxact:right>
            </ipxact:vector>
          </ipxact:vectors>
          <ipxact:value>comp_dual_mode?4'hf:4'h0</ipxact:value>
        </ipxact:moduleParameter>
      </ipxact:moduleParameters>
      <ipxact:fileSetRef>
        <ipxact:localName>VerilogFiles</ipxact:localName>
      </ipxact:fileSetRef>
    </ipxact:componentInstantiation>
    <ipxact:componentInstantiation>
      <ipxact:name>TLMModel</ipxact:name>
      <ipxact:language>SystemC</ipxact:language>
      <ipxact:moduleName>sample</ipxact:moduleName>
      <ipxact:moduleParameters>
        <ipxact:moduleParameter type="bit">
          <ipxact:name>dual_mode</ipxact:name>
          <ipxact:value>comp_dual_mode</ipxact:value>
        </ipxact:moduleParameter>
        <ipxact:moduleParameter parameterId="vdp" resolve="generated"
type="bit">
          <ipxact:name>view_dependent_param</ipxact:name>
          <ipxact:value>0</ipxact:value>
        </ipxact:moduleParameter>
      </ipxact:moduleParameters>
      <ipxact:fileSetRef>
        <ipxact:localName>SystemCFiles</ipxact:localName>
      </ipxact:fileSetRef>
    </ipxact:componentInstantiation>
  </ipxact:instantiations>
  <ipxact:ports>
    <!-- LINK: port: see 6.12.7, Component ports -->
    <ipxact:port>
      <ipxact:name>slv_data</ipxact:name>
      <!-- LINK: wire: see 6.12.8, Component wire ports -->
      <ipxact:wire>
        <ipxact:direction>in</ipxact:direction>
        <ipxact:vectors>
          <ipxact:vector>
            <ipxact:left>15</ipxact:left><ipxact:right>0</ipxact:right>
          </ipxact:vector>
        </ipxact:vectors>
        <ipxact:wireTypeDefs>
          <!-- LINK: wireTypeDef: see 6.12.9, Component wireTypeDef -->
          <ipxact:wireTypeDef>
            <ipxact:typeName>logic</ipxact:typeName>
            <ipxact:viewRef>RTLview</ipxact:viewRef>
          </ipxact:wireTypeDef>
        </ipxact:wireTypeDefs>
        <!-- Default upper 8 bits since the interface only uses 8 bits -->
        <ipxact:drivers>
          <!-- LINK: driver: see 6.12.10, Component driver -->
          <ipxact:driver>
            <ipxact:range>
              <ipxact:left>15</ipxact:left><ipxact:right>8</ipxact:right>
            </ipxact:range>
            <ipxact:defaultValue>8'h0</ipxact:defaultValue>
```


Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
            </ipxact:driver>
          </ipxact:drivers>
        </ipxact:wire>
      </ipxact:port>
      <ipxact:port>
        <ipxact:name>mst_data</ipxact:name>
        <ipxact:wire>
          <ipxact:direction>out</ipxact:direction>
          <ipxact:vectors>
            <ipxact:vector>
              <ipxact:left>7</ipxact:left><ipxact:right>0</ipxact:right>
            </ipxact:vector>
          </ipxact:vectors>
          <ipxact:wireTypeDefs>
            <ipxact:wireTypeDef>
              <ipxact:typeName>logic</ipxact:typeName>
              <ipxact:viewRef>RTLview</ipxact:viewRef>
            </ipxact:wireTypeDef>
          </ipxact:wireTypeDefs>
        </ipxact:wire>
      </ipxact:port>
      <ipxact:port>
        <ipxact:name>slv_addr</ipxact:name>
        <ipxact:wire>
          <ipxact:direction>in</ipxact:direction>
          <ipxact:vectors>
            <ipxact:vector>
              <ipxact:left>7</ipxact:left><ipxact:right>0</ipxact:right>
            </ipxact:vector>
          </ipxact:vectors>
          <ipxact:wireTypeDefs>
            <ipxact:wireTypeDef>
              <ipxact:typeName>logic</ipxact:typeName>
              <ipxact:viewRef>RTLview</ipxact:viewRef>
            </ipxact:wireTypeDef>
          </ipxact:wireTypeDefs>
          <ipxact:constraintSets>
<!-- LINK: constraintSet: see 6.12.13, Component wire port constraints -->
            <ipxact:constraintSet>
             <!-- Port driven by high strength sequential cell for synthesis -->
              <ipxact:driveConstraint>
              <!-- LINK: cellSpecification: see 6.12.15, Load and drive constraint
   cell specification -->
                <ipxact:cellSpecification cellStrength="high">
                  <ipxact:cellClass>sequential</ipxact:cellClass>
                </ipxact:cellSpecification>
              </ipxact:driveConstraint>
              <!-- Port timing requirements for synthesis -->
            <!-- LINK: timingConstraint: see 6.12.14, Port timing constraints -->
              <ipxact:timingConstraint clockName="clk">60</
   ipxact:timingConstraint>
            </ipxact:constraintSet>
          </ipxact:constraintSets>
        </ipxact:wire>
      </ipxact:port>
      <ipxact:port>
        <ipxact:name>mst_addr</ipxact:name>
        <ipxact:wire>
          <ipxact:direction>out</ipxact:direction>
```

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
          <ipxact:vectors>
            <ipxact:vector>
              <ipxact:left>7</ipxact:left><ipxact:right>0</ipxact:right>
            </ipxact:vector>
          </ipxact:vectors>
          <ipxact:wireTypeDefs>
            <ipxact:wireTypeDef>
              <ipxact:typeName>logic</ipxact:typeName>
              <ipxact:viewRef>RTLview</ipxact:viewRef>
            </ipxact:wireTypeDef>
          </ipxact:wireTypeDefs>
          <ipxact:constraintSets>
            <ipxact:constraintSet>
             <!-- Port drives 2 high strength sequential cells for synthesis -->
              <ipxact:loadConstraint>
                <ipxact:cellSpecification cellStrength="high">
                  <ipxact:cellClass>sequential</ipxact:cellClass>
                </ipxact:cellSpecification>
                <ipxact:count>2</ipxact:count>
              </ipxact:loadConstraint>
              <!-- Port timing requirements for synthesis -->
              <ipxact:timingConstraint clockName="clk">40</
ipxact:timingConstraint>
            </ipxact:constraintSet>
          </ipxact:constraintSets>
        </ipxact:wire>
      </ipxact:port>
      <!-- parity port exists in RTL view with default data type -->
      <ipxact:port>
        <ipxact:name>slv_parity</ipxact:name>
        <ipxact:wire>
          <ipxact:direction>in</ipxact:direction>
          <ipxact:wireTypeDefs>
            <ipxact:wireTypeDef>
              <ipxact:viewRef>RTLview</ipxact:viewRef>
            </ipxact:wireTypeDef>
          </ipxact:wireTypeDefs>
        </ipxact:wire>
      </ipxact:port>
      <!-- clock port exists in RTL view with default data type -->
      <ipxact:port>
        <ipxact:name>clk</ipxact:name>
        <ipxact:wire>
          <ipxact:direction>in</ipxact:direction>
          <ipxact:wireTypeDefs>
            <ipxact:wireTypeDef>
              <ipxact:viewRef>RTLview</ipxact:viewRef>
            </ipxact:wireTypeDef>
          </ipxact:wireTypeDefs>
          <ipxact:drivers>
            <ipxact:driver>
             <!-- LINK: clockDriver: see 6.12.11, Component driver/clockDriver -->
              <ipxact:clockDriver>
                <ipxact:clockPeriod>5</ipxact:clockPeriod>
                <ipxact:clockPulseOffset>2.5</ipxact:clockPulseOffset>
                <ipxact:clockPulseValue>1</ipxact:clockPulseValue>
                <ipxact:clockPulseDuration>2.5</ipxact:clockPulseDuration>
              </ipxact:clockDriver>
            </ipxact:driver>
```


Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
          </ipxact:drivers>
        </ipxact:wire>
      </ipxact:port>
      <!-- reset port exists in RTL view with default data type -->
      <ipxact:port>
        <ipxact:name>reset</ipxact:name>
        <ipxact:wire>
          <ipxact:direction>in</ipxact:direction>
          <ipxact:wireTypeDefs>
            <ipxact:wireTypeDef>
              <ipxact:viewRef>RTLview</ipxact:viewRef>
            </ipxact:wireTypeDef>
          </ipxact:wireTypeDefs>
          <ipxact:drivers>
            <ipxact:driver>
              <ipxact:singleShotDriver>
                <ipxact:singleShotOffset>2</ipxact:singleShotOffset>
                <ipxact:singleShotValue>1</ipxact:singleShotValue>
                <ipxact:singleShotDuration>5</ipxact:singleShotDuration>
              </ipxact:singleShotDriver>
            </ipxact:driver>
          </ipxact:drivers>
        </ipxact:wire>
      </ipxact:port>
      <!-- status port exists in all views with default data type -->
      <ipxact:port>
        <ipxact:name>status</ipxact:name>
        <ipxact:wire>
          <ipxact:direction>out</ipxact:direction>
          <ipxact:vectors>
            <ipxact:vector>
              <ipxact:left>7</ipxact:left><ipxact:right>0</ipxact:right>
            </ipxact:vector>
          </ipxact:vectors>
        </ipxact:wire>
      </ipxact:port>
      <!-- anotherPort port exists in all views with default data type -->
      <ipxact:port>
        <ipxact:name>anotherPort</ipxact:name>
        <ipxact:wire>
          <ipxact:direction>in</ipxact:direction>
          <ipxact:vectors>
            <ipxact:vector>
              <ipxact:left>3</ipxact:left><ipxact:right>0</ipxact:right>
            </ipxact:vector>
          </ipxact:vectors>
        </ipxact:wire>
      </ipxact:port>
      <!-- slv_transaction port exists in TLM view only -->
      <ipxact:port>
        <ipxact:name>slv_transaction</ipxact:name>
<!-- LINK: transactional: see 6.12.17, Component transactional port type -->
        <ipxact:transactional>
          <ipxact:initiative>requires</ipxact:initiative>
          <ipxact:transTypeDefs>
            <!-- LINK: transTypeDef: see 6.12.19, Component transactional port
   type definition -->
            <ipxact:transTypeDef>
              <ipxact:typeName>sampleTLMtype</ipxact:typeName>
```

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
                <ipxact:viewRef>TLMview</ipxact:viewRef>
              </ipxact:transTypeDef>
            </ipxact:transTypeDefs>
          </ipxact:transactional>
        </ipxact:port>
        <!-- mst_transaction port exists in TLM view only -->
        <ipxact:port>
          <ipxact:name>mst_transaction</ipxact:name>
          <ipxact:transactional>
            <ipxact:initiative>provides</ipxact:initiative>
            <ipxact:transTypeDefs>
              <ipxact:transTypeDef>
                <ipxact:typeName>sampleTLMtype</ipxact:typeName>
                <ipxact:viewRef>TLMview</ipxact:viewRef>
              </ipxact:transTypeDef>
            </ipxact:transTypeDefs>
          </ipxact:transactional>
        </ipxact:port>
      </ipxact:ports>
    </ipxact:model>
    <ipxact:componentGenerators>
      <!-- LINK: componentGenerator: see 6.13, Component generators -->
      <ipxact:componentGenerator>
        <ipxact:name>SimpleComponentGenerator</ipxact:name>
        <ipxact:description>Simple TGI based component generator</
      ipxact:description>
        <ipxact:phase>1</ipxact:phase>
        <ipxact:parameters>
          <ipxact:parameter parameterId="genParm1" prompt="Parm 1"
      type="shortint" resolve="user">
            <ipxact:name>genParm1</ipxact:name>
            <ipxact:description>First generator parameter.</ipxact:description>
            <ipxact:value>1</ipxact:value>
          </ipxact:parameter>
          <ipxact:parameter parameterId="genParm2" prompt="Parm 2"
      type="shortint" resolve="user">
            <ipxact:name>genParm2</ipxact:name>
           <ipxact:description>Second generator parameter.</ipxact:description>
            <ipxact:value>2</ipxact:value>
          </ipxact:parameter>
        </ipxact:parameters>
        <ipxact:apiType>TGI_2014_BASE</ipxact:apiType>
        <!-- By convention, generator invoked and passed parameters as follows:
      -->
        <!-- generator.sh -genParm1 <value> -genParm2 <value> -->
        <ipxact:generatorExe>../bin/generator.sh</ipxact:generatorExe>
      </ipxact:componentGenerator>
    </ipxact:componentGenerators>
    <!-- LINK: choices: see 6.14, Choices -->
    <ipxact:choices>
      <ipxact:choice>
        <ipxact:name>bitsize</ipxact:name>
        <ipxact:enumeration text="32 bits">32</ipxact:enumeration>
        <ipxact:enumeration text="64 bits">64</ipxact:enumeration>
      </ipxact:choice>
    </ipxact:choices>
    <ipxact:fileSets>
      <!-- LINK: fileSet: see 6.15.1, File sets -->
      <ipxact:fileSet>
```


Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
        <ipxact:name>VerilogFiles</ipxact:name>
        <!-- LINK: file: see 6.15.2, file -->
        <ipxact:file>
          <ipxact:name>../src/component.v</ipxact:name>
          <ipxact:fileType>verilogSource</ipxact:fileType>
          <ipxact:isStructural>true</ipxact:isStructural>
        </ipxact:file>
        <ipxact:file>
          <ipxact:name>../src/component.v</ipxact:name>
          <ipxact:fileType>verilogSource</ipxact:fileType>
        </ipxact:file>
      </ipxact:fileSet>
      <ipxact:fileSet>
        <ipxact:name>SystemCFiles</ipxact:name>
        <ipxact:file>
          <ipxact:name>../src/component.C</ipxact:name>
          <ipxact:fileType>systemCSource-2.2</ipxact:fileType>
        </ipxact:file>
      </ipxact:fileSet>
    </ipxact:fileSets>
    <ipxact:whiteboxElements>
  <!-- LINK: whiteboxElement: see 6.16, White box elements -->
      <ipxact:whiteboxElement>
        <ipxact:name>ImportantInternalSignal</ipxact:name>
        <ipxact:description>Defines access point for forcing sim values</
     ipxact:description>
        <ipxact:whiteboxType>signal</ipxact:whiteboxType>
        <ipxact:driveable>true</ipxact:driveable>
      </ipxact:whiteboxElement>
    </ipxact:whiteboxElements>
    <ipxact:cpus>
      <!-- LINK: cpu: see 6.18, CPUs -->
      <ipxact:cpu>
        <ipxact:name>cpuDefn</ipxact:name>
        <ipxact:addressSpaceRef addressSpaceRef="simpleAddressSpace"/>
      </ipxact:cpu>
    </ipxact:cpus>
    <ipxact:otherClockDrivers>
      <!-- LINK: otherClockDriver: see 6.12.16, Other clock drivers -->
      <ipxact:otherClockDriver clockName="virtualClock1">
        <ipxact:clockPeriod>10</ipxact:clockPeriod>
        <ipxact:clockPulseOffset>5</ipxact:clockPulseOffset>
        <ipxact:clockPulseValue>1</ipxact:clockPulseValue>
        <ipxact:clockPulseDuration>5</ipxact:clockPulseDuration>
      </ipxact:otherClockDriver>
    </ipxact:otherClockDrivers>
    <ipxact:resetTypes>
      <!-- TODO: MISSING definition of resetType in document -->
      <ipxact:resetType>
        <!-- LINK: resetType: see 6.19, Reset type -->
        <ipxact:name>SOFT</ipxact:name>
        <ipxact:displayName>Soft Reset</ipxact:displayName>
      </ipxact:resetType>
    </ipxact:resetTypes>
    <ipxact:description>Example component used in the IP-XACT standard.</
     ipxact:description>
    <!-- LINK: parameters: see C.18, parameters -->
    <ipxact:parameters>
      <!-- LINK: parameters: see C.19, attributes -->
```

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
      <ipxact:parameter parameterId="TLMModelsAvailable" prompt="TLM Models
      Available" type="bit" resolve="user">
        <ipxact:name>TLMModelsAvailable</ipxact:name>
        <ipxact:displayName>TLM Models Available</ipxact:displayName>
       <ipxact:description>Set to true if TLM simulation models are available.</
      ipxact:description>
        <ipxact:value>0</ipxact:value>
      </ipxact:parameter>
      <ipxact:parameter parameterId="comp_dual_mode" prompt="Dual Mode
      Supported" type="bit" resolve="user">
        <ipxact:name>comp_dual_mode</ipxact:name>
        <ipxact:description>Indicates dual mode support is desired.</
      ipxact:description>
        <ipxact:value>1</ipxact:value>
      </ipxact:parameter>
      <!-- Parameter leveraging an enumeration 'choice' -->
      <ipxact:parameter parameterId="addrBits" choiceRef="bitsize"
      prompt="Address Bits" type="shortint" resolve="user">
        <ipxact:name>addrBits</ipxact:name>
        <ipxact:value>32</ipxact:value>
      </ipxact:parameter>
    </ipxact:parameters>
    <!-- LINK: assertions: see C.2, assertions -->
    <ipxact:assertions>
      <ipxact:assertion>
        <ipxact:name>ArbitraryAssertion1</ipxact:name>
       <ipxact:description>Arbitrary assertion to show syntax. Must evaluate to
      true.</ipxact:description>
        <ipxact:assert>TLMModelsAvailable || !comp_dual_mode</ipxact:assert>
      </ipxact:assertion>
    </ipxact:assertions>
    <!-- LINK: vendorExtensions: see C.24, vendorExtensions -->
    <ipxact:vendorExtensions>
      <!-- Simple example showing elements from a separate name space added via
      vendorExtensions. -->
      <!-- The namespace must be defined (xmlns:<ns>="<URL>") at top or within
      first usage. -->
      <sampleNS:extraElement xmlns:sampleNS="http://www.nosuchURL.org/Schema">
        <sampleNS:anotherElement sampleNS:attributeName="XYZ"/>
      </sampleNS:extraElement>
    </ipxact:vendorExtensions>
  </ipxact:component>
```

I.7 design

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
// Example design used to show schema elements defined by the
// IP-XACT standard. Links within this file refer to schema figures in
// the standard definition document.
-->
<!-- LINK: design: see 7.1, Design -->
<ipxact:design xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/2.0"
   xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/2.0/
   index.xsd">
  <ipxact:vendor>accellera.org</ipxact:vendor>
```


Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
<ipxact:library>Sample</ipxact:library>
<ipxact:name>SampleDesign</ipxact:name>
<ipxact:version>1.0</ipxact:version>
<!-- Two RTL instances (U1, U2), and 1 TLM instance (U3) -->
<ipxact:componentInstances>
  <!-- LINK: componentInstance: see 7.2, Design component instances -->
  <ipxact:componentInstance>
    <ipxact:instanceName>U1</ipxact:instanceName>
    <ipxact:componentRef vendor="accellera.org" library="Sample"
 name="SampleComponent" version="1.0">
      <ipxact:configurableElementValues>
        <ipxact:configurableElementValue referenceId="comp_dual_mode">0</
 ipxact:configurableElementValue>
      </ipxact:configurableElementValues>
    </ipxact:componentRef>
  </ipxact:componentInstance>
  <ipxact:componentInstance>
    <ipxact:instanceName>U2</ipxact:instanceName>
    <ipxact:componentRef vendor="accellera.org" library="Sample"
 name="SampleComponent" version="1.0">
    </ipxact:componentRef>
  </ipxact:componentInstance>
  <ipxact:componentInstance>
    <ipxact:instanceName>U3</ipxact:instanceName>
    <ipxact:componentRef vendor="accellera.org" library="Sample"
 name="SampleComponent" version="1.0">
    </ipxact:componentRef>
  </ipxact:componentInstance>
</ipxact:componentInstances>
<ipxact:interconnections>
  <!-- Simple master to slave interface connection -->
  <!-- LINK: interconnection: see 7.3.1, interconnection -->
  <ipxact:interconnection>
    <ipxact:name>TwoIntfs</ipxact:name>
   <!-- LINK: activeInterface: see 7.4, Active, hierarchical, monitored, and
 monitor interfaces -->
    <ipxact:activeInterface componentRef="U1" busRef="Master"/>
    <ipxact:activeInterface componentRef="U2" busRef="Slave"/>
  </ipxact:interconnection>
  <!-- Broadcast interface from a slave (one to many connection) -->
  <ipxact:interconnection>
    <ipxact:name>BroadcastConnection</ipxact:name>
    <ipxact:activeInterface componentRef="U1" busRef="Slave"/>
    <ipxact:hierInterface busRef="Slave0"/>
    <ipxact:hierInterface busRef="Slave1"/>
  </ipxact:interconnection>
 <!-- Export Master interface -- will be used for TLM to RTL conversion -->
  <ipxact:interconnection>
    <ipxact:name>ExportTLM</ipxact:name>
    <ipxact:activeInterface componentRef="U3" busRef="Master"/>
    <ipxact:hierInterface busRef="TLMMaster"/>
  </ipxact:interconnection>
  <!-- LINK: monitorInterconnection: 7.3.2 monitorInterconnection -->
  <!-- Monitor interface connection -->
  <ipxact:monitorInterconnection>
    <ipxact:name>MonitorConnection</ipxact:name>
    <ipxact:monitoredActiveInterface componentRef="U2" busRef="Slave"/>
    <ipxact:monitorInterface componentRef="U2" busRef="Monitor"/>
  </ipxact:monitorInterconnection>
```

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
    </ipxact:interconnections>
    <ipxact:adHocConnections>
      <!-- Simple two pin connection. Range not mentioned on anotherPort -->
      <!-- which implies all bits are connected -->
      <!-- LINK: adHocConnection: see 7.5, Design ad hoc connections -->
      <ipxact:adHocConnection>
        <ipxact:name>SimpleWire</ipxact:name>
        <ipxact:portReferences>
          <!-- LINK: internalPortReference: see 7.6, Port references -->
        <ipxact:internalPortReference componentRef="U1" portRef="anotherPort"/>
          <ipxact:internalPortReference componentRef="U2" portRef="status">
            <ipxact:partSelect>
              <ipxact:range><ipxact:left>7</ipxact:left><ipxact:right>4</
    ipxact:right></ipxact:range>
            </ipxact:partSelect>
          </ipxact:internalPortReference>
        </ipxact:portReferences>
      </ipxact:adHocConnection>
      <!-- Tie input bits to a constant -->
      <ipxact:adHocConnection>
        <ipxact:name>TieOff</ipxact:name>
        <ipxact:tiedValue>3'h2</ipxact:tiedValue>
        <ipxact:portReferences>
         <ipxact:internalPortReference componentRef="U2" portRef="anotherPort">
            <ipxact:partSelect>
              <ipxact:range><ipxact:left>2</ipxact:left><ipxact:right>0</
    ipxact:right></ipxact:range>
            </ipxact:partSelect>
          </ipxact:internalPortReference>
        </ipxact:portReferences>
      </ipxact:adHocConnection>
      <!-- Tie output bits to open -->
      <ipxact:adHocConnection>
        <ipxact:name>TieOpen</ipxact:name>
        <ipxact:tiedValue>open</ipxact:tiedValue>
        <ipxact:portReferences>
          <ipxact:internalPortReference componentRef="U1" portRef="status"/>
        </ipxact:portReferences>
      </ipxact:adHocConnection>
    </ipxact:adHocConnections>
  </ipxact:design>
```

I.8 designConfiguration

```
  <?xml version="1.0" encoding="UTF-8"?>
  <!--
  // Example designConfiguration used to show schema elements defined by the
  // IP-XACT standard. Links within this file refer to schema figures in
  // the standard definition document.
  -->
  <!-- LINK: designConfiguration: see 10.2, designConfiguration -->
  <ipxact:designConfiguration xmlns:xsi="http://www.w3.org/2001/XMLSchema-
     instance" xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/2.0"
     xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/2.0/
     index.xsd">
    <ipxact:vendor>accellera.org</ipxact:vendor>
    <ipxact:library>Sample</ipxact:library>
```


Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
  <ipxact:name>SampleDesignConfiguration</ipxact:name>
  <ipxact:version>1.0</ipxact:version>
 <ipxact:generatorChainConfiguration vendor="accellera.org" library="Sample"
   name="SampleGeneratorChain" version="1.0">
    <ipxact:configurableElementValues>
      <ipxact:configurableElementValue referenceId="genParm1">7</
   ipxact:configurableElementValue>
    </ipxact:configurableElementValues>
 </ipxact:generatorChainConfiguration>
 <!-- LINK: interconnectionConfiguration: see 10.3,
   interconnectionConfiguration -->
 <ipxact:interconnectionConfiguration>
   <ipxact:interconnectionRef>ExportTLM</ipxact:interconnectionRef>
   <ipxact:abstractorInstances>
     <!-- LINK: abstractorInstance: see 10.4, abstractor instance -->
     <ipxact:abstractorInstance>
       <ipxact:instanceName>U_tlm2rtl</ipxact:instanceName>
   <ipxact:abstractorRef vendor="accellera.org" library="Sample"
   name="SampleAbstractor" version="1.0"/>
       <ipxact:viewName>default</ipxact:viewName>
     </ipxact:abstractorInstance>
   </ipxact:abstractorInstances>
 </ipxact:interconnectionConfiguration>
 <!-- Select RTL for U1, U2, and TLM for U3 -->
 <!-- LINK: viewConfiguration: see 10.5, viewConfiguration -->
 <ipxact:viewConfiguration>
   <ipxact:instanceName>U1</ipxact:instanceName>
   <ipxact:view viewRef="RTLview"/>
 </ipxact:viewConfiguration>
 <ipxact:viewConfiguration>
   <ipxact:instanceName>U2</ipxact:instanceName>
   <ipxact:view viewRef="RTLview"/>
 </ipxact:viewConfiguration>
 <ipxact:viewConfiguration>
   <ipxact:instanceName>U3</ipxact:instanceName>
   <ipxact:view viewRef="TLMview">
     <ipxact:configurableElementValues>
       <ipxact:configurableElementValue referenceId="vdp">1</
   ipxact:configurableElementValue>
     </ipxact:configurableElementValues>
   </ipxact:view>
 </ipxact:viewConfiguration>
</ipxact:designConfiguration>
```

I.9 generatorChain

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
// Example generatorChain used to show schema elements defined by the
// IP-XACT standard. Links within this file refer to schema figures in
// the standard definition document
-->
<!-- LINK: generatorChain: see 9.1, generatorChain -->
<ipxact:generatorChain xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ipxact="http://www.accellera.org/XMLSchema/IPXACT/2.0"
    xsi:schemaLocation="http://www.accellera.org/XMLSchema/IPXACT/2.0/
   index.xsd">
```

Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
<ipxact:vendor>accellera.org</ipxact:vendor>
<ipxact:library>Sample</ipxact:library>
<ipxact:name>SampleGeneratorChain</ipxact:name>
<ipxact:version>1.0</ipxact:version>
<!-- Include generators with the indicated group names -->
<!-- LINK: generatorChainSelector: see 9.2, generatorChainSelector -->
<ipxact:generatorChainSelector>
  <ipxact:groupSelector multipleGroupSelectionOperator="and">
    <ipxact:name>HDLAnalysis</ipxact:name>
    <ipxact:name>HDLElaboration</ipxact:name>
  </ipxact:groupSelector>
</ipxact:generatorChainSelector>
<!-- Include generator with the indicated VLNV -->
<ipxact:generatorChainSelector>
  <ipxact:generatorChainRef vendor="accellera.org" library="Sample"
                      name="SampleGeneratorChainForReference"
                      version="1.0"/>
</ipxact:generatorChainSelector>
<!-- Include component generator with the indicated group name -->
<!-- LINK: componentGeneratorSelector: see 9.3, generatorChain component
 selector -->
<ipxact:componentGeneratorSelector>
  <ipxact:groupSelector>
    <ipxact:name>DocGeneration</ipxact:name>
  </ipxact:groupSelector>
</ipxact:componentGeneratorSelector>
<!-- Define a generator explicitly within the generator chain -->
<!-- LINK: generator: see 9.4, generatorChain generator -->
<ipxact:generator hidden="false">
  <ipxact:name>myGenerator</ipxact:name>
  <ipxact:displayName>My Generator</ipxact:displayName>
  <ipxact:description>This is my generator</ipxact:description>
  <ipxact:phase>100.0</ipxact:phase>
  <ipxact:parameters>
   <ipxact:parameter parameterId="genParm1" prompt="Parm 1" type="shortint"
 resolve="user">
      <ipxact:name>genParm1</ipxact:name>
      <ipxact:description>First generator parameter.</ipxact:description>
      <ipxact:value>1</ipxact:value>
    </ipxact:parameter>
  </ipxact:parameters>
  <ipxact:apiType>TGI_2014_BASE</ipxact:apiType>
  <ipxact:transportMethods>
    <ipxact:transportMethod>file</ipxact:transportMethod>
  </ipxact:transportMethods>
  <ipxact:generatorExe>../bin/run.sh</ipxact:generatorExe>
</ipxact:generator>
<ipxact:chainGroup>implementation</ipxact:chainGroup>
<ipxact:displayName>Sample Generator Chain</ipxact:displayName>
<ipxact:description>Example generator chain used in the IP-XACT standard.</
 ipxact:description>
```


Copyright © 2014 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.